NOVEL PUSH-MODEL FRAMEWORK FOR REALIZING MULTI-SERVICE MPEG4 STREAMS IN BROADCAST APPLICATIONS

by

AMR M. MOHAMED
B.Sc. Cairo University, Egypt, 1993

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF BRITISH COLUMBIA
September 2001
© Amr M. S. Mohamed, 2001
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada

Date Oct. 15, 2001

DE-6 (2/88)
Abstract

In this thesis, a new broadcast framework is proposed for realizing the MPEG4 media content. This framework defines some possible techniques that may be used for realizing MPEG4 streams in broadcast applications. The thesis proposes solutions to some of the problems associated with broadcasting MPEG-4 media streams regarding service advertisement, media streams management, transport-level streams synchronization, and bandwidth control. To verify the functionality of this framework, a client/server software has been developed, which includes two main modules for service advertisement and MPEG-4 media streams management and delivery. The software architecture adopts the push model for data distribution in all modules to minimize the information delivered on the upstream network media. The service advertisement module adopts SAP/SDP protocol stack for announcing services’ description. The media streams management module adopts the MPEG-4 DMIF standard for media streams management. The framework comprises the main portions for broadcasting MPEG-4 streams including data channels multiplexing to group the data channels’ flows to one or more network flows, and client random access to allow clients to access the service at any time during the service presentation. The implementation uses MPEG-4 IM1 reference software for elementary streams creation, decoding, and synchronization. The implementation tackles some issues related to the broadcast data delivery. These issues include transport-level data channels synchronization for synchronizing service’s audio-visual objects before data transport. The implementation also includes two algorithms for server’s bandwidth control. ServicePacketDrop algorithm controls the server’s bandwidth on the service level while ObjectPacketDrop control the server’s bandwidth on the object level.
# Table Of Contents

Abstract ......................................................................................................................... ii

List Of Tables .............................................................................................................. vi

List Of Figures ............................................................................................................. vii

Acknowledgment ......................................................................................................... ix

Chapter 1  Introduction ................................................................................................. 1

1.1 Motivation ................................................................................................................ 1

1.2 Previous and Related work ..................................................................................... 1

1.3 Objectives and contributions ................................................................................. 5

1.4 Framework benefits ............................................................................................... 6

1.5 Scope and outline .................................................................................................... 7

Chapter 2  MPEG-4 Broadcast Systems ....................................................................... 10

2.1 MPEG4 requirements ............................................................................................ 11

2.1.1 Traditional MPEG Systems requirements ......................................................... 11

2.1.2 Specific MPEG-4 Systems requirements ............................................................. 13

2.2 MPEG4 Terminal Architecture .............................................................................. 15

2.3 MPEG4 systems tools .......................................................................................... 21

2.3.1 System Management ........................................................................................ 21

2.3.2 Object Presentation (composition): BIFS ............................................................ 23

2.3.3 Timing and synchronization .............................................................................. 25

2.3.4 MPEG-4 delivery ............................................................................................. 26

2.4 Live Media Broadcast ......................................................................................... 30

2.4.1 Features and Benefits ....................................................................................... 31
List Of Tables

Table 3.1  Session description protocol fields .................................................. 36
Table 4.1  ObjectPacketDrop flow .................................................................. 65
Table 5.1  List of parameters ............................................................................ 70
**List Of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>MPEG4 systems architecture</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Object descriptors linking scene description to elementary streams</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>DMIF communication architecture</td>
<td>29</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>DMIF addresses the delivery integration of three major technologies</td>
<td>30</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Live broadcast distribution using multiple transport scenarios</td>
<td>32</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Broadcast System architecture</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Publisher-subscriber model for information distribution</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>SAP Header fields</td>
<td>38</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Client subscription PDU</td>
<td>40</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Clients grouping for IP multicast</td>
<td>41</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>DMIF data channels</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>MPEG-4 DMIF broadcast model for client random access</td>
<td>44</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Transport Packet Header Format</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Module Diagram of Broadcast service advertisement</td>
<td>52</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Architecture of Broadcast DMIF Terminal</td>
<td>53</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Data Channels synchronization</td>
<td>58</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>ServicePacketDrop Algorithm</td>
<td>61</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Data channels sorting</td>
<td>63</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>ObjectPacketDrop Algorithm</td>
<td>64</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Multiple service layer</td>
<td>66</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>Multiple service layer modules</td>
<td>67</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Instant bit rate for “Linda” video stream</td>
<td>71</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Decoding time stamps and Receiving time stamps</td>
<td>73</td>
</tr>
</tbody>
</table>
Figure 5.3 average bit rate for "Linda" audio-visual stream ........................................ 75
Figure 5.4 Instant average bit rate for 16 services ......................................................... 76
Figure 5.5 Average overhead over time ........................................................................... 77
Figure 5.6 Client startup random variable ...................................................................... 80
Figure 5.7 Average overhead versus number of services ................................................. 80
Figure 5.8 Data Channels' Instant bit-rate for "Linda" audio-visual stream ....................... 81
Figure 5.9 Service instant/maximum bit-rate for "Linda" audio-visual stream with 2 seconds window size ................................................................. 82
Figure 5.10 "Linda" audio-visual stream with ServicePacketDrop (1) ................................. 83
Figure 5.11 "Linda" stream a) No Bandwidth Control b) with 100Kbps ServicePacketDrop ..................................................................................................................... 83
Figure 5.12 "Linda" audio-visual stream with ServicePacketDrop (2) ................................. 84
Figure 5.13 8-Object stream with ServicePacketDrop Algorithm (1) ................................. 84
Figure 5.14 8-Object visual stream with no bandwidth control ....................................... 85
Figure 5.15 8-Object stream with ServicePacketDrop Algorithm (2) ................................. 85
Figure 5.16 8-Object stream with 800Kbps ServicePacketDrop ...................................... 86
Figure 5.17 Timing for server's bit rates calculations ......................................................... 87
Figure 5.18 8-object stream using ObjectPacketDrop algorithm (1) ................................. 88
Figure 5.19 8-object stream using ObjectPacketDrop algorithm (2) ................................. 89
Figure 5.20 8-Object stream with 800Kbps ObjectPacketDrop ........................................ 89
Figure A.1 IM1 main system modules ............................................................................... 98
Figure D.2 Service description based on SDP information .............................................. 108
Figure D.3 Server user interface .................................................................................... 109
Figure D.4 Client user interface ..................................................................................... 110
Acknowledgment

I would like to express my gratitude to my research advisor, Dr. Hussein M. Alnuweiri, for his guidance throughout this research work. My conversations with him were always inspirational, informative, and entertaining.

I would like to thank my wife for her significant effort to edit the thesis and for her patience and continuous encouragement. I would like to thank my baby daughter for making my life pleasant, objective and enjoyable. Most important acknowledgement goes to my Parents, and my brother for their remote support and guidance. I would like to thank my true friends Ahmed Ismail, Ayman Malek, Tamer Khattab and their families for their sincere support.

I would like to thank my research colleagues Yasser Pourmohammadi and Kambiz Asrar for their constructive cooperation, knowledge and resources exchange throughout this research work.

Lastly, and foremost, I thank God for blessing me with knowledge and energy to accomplish this goal, and I thank him for making all this possible.
Chapter 1 Introduction

1.1 Motivation

Several multimedia applications, including online presentation, distant learning, and satellite TV over Packet networks, play significant part in growing the demand for broadcast scenario as a potential delivery technology for multimedia content. Therefore, the demand for broadcast delivery technology is growing hand in hand with – if not over - the demand for other multimedia delivery scenarios including remote and local storage media access. Nevertheless, the amount of research work going in the field of multimedia broadcast delivery is inferior compared to the emerging research going for other scenarios. The lack of standardization, which provides complete broadcast solution, may be responsible for such unbalance between demand and research work. This thesis focuses on the broadcast scenario for media delivery as an ultimate scalable way for multimedia distribution and proposes a solution for building a broadcast framework for streaming object-based presentations. The thesis discusses the main aspects of building a broadcast streaming solution and provides techniques and algorithms to enhance the performance of object-based presentations in terms of media synchronization and bandwidth utilization.

1.2 Previous and Related work

Relative to traditional video broadcast, this thesis introduces a sophisticated video broadcast/multicast architecture based on object technology and push communication model. Up until now, very little work has been done in this area, even though object-based video and push communication models are well known concepts [1][2]. Previous work on video
broadcast/multicast is mostly based on traditional frame-based video content and basic audio-video synchronization. In this section we briefly review widely deployed technologies for video delivery.

Commercial software multimedia clients such as Realplayer from Real Networks (http://www.real.com), MediaPlayer from Microsoft (http://www.microsoft.com), and Quicktime from Apple Computer (http://www.apple.com/quicktime) usually use real-time streaming technology to retrieve multimedia contents by fetching information from servers in a pull manner. Clients perform this process by maintaining connections with the server and controlling the data sending throughout the service presentation. This requires the server to create and maintain one service per each client's request and to maintain connections with all the clients throughout the entire service presentation. This limits the server's capability to support only a limited number of customers by allocating bandwidth to individual clients' requests.

Video Broadcast is another methodology for media distribution that does not require the server to maintain connections with the clients. This allows streaming server to accommodate more clients without incurring almost any overhead for each client. This scenario saves the server bandwidth and enhances the scalability of the server through supporting a larger number of clients. The broadcast scenario, however, requires special synchronization mechanisms during the multimedia content delivery to allow clients to tune in at any time during the service presentation. Rate control or traffic shaping mechanisms may not work with this broadcast scenario because the mechanism may allow more
information to be sent at the beginning of the presentation and that affects the clients listening to the service from a middle point during the service presentation.

Traditional broadcast multimedia applications (e.g. Broadcast Real System Server 8 from Real Networks, and Tibco's Rendezvous www.rv.tibco.com) compose the multimedia scene into a set of rectangular temporal frames before being compressed or transmitted through the broadcast network. For example, ISO/IEC 13818-6:1998/Amd part 1 and 2 standards define the broadcast of MPEG-2 packetized elementary streams (PESs) over broadcast networks. Such applications are called frame-based applications. Frame-based applications do not benefit from the semantics of the multimedia scene to provide higher quality coding and higher network and server utilization during media delivery.

Recent advances in image and video encoding and representation techniques have played an important role in encoding and representation of audio-visual scenes with semantically meaningful objects. This concept is called object-based media representation, and it forms the basis of contemporary multimedia coding and delivery standards, such as MPEG-4 [2]. The object-based concept enhances the scene interaction while providing more efficient media encoding and transport by optimizing the encoding and network recourses based on each object's characteristics. The object-based application composes the scene into a set of audio-visual objects that are encoded and transmitted collaboratively. While this concept provides more robust, higher quality encoding, and adds more flexibility for media delivery, it creates more complexities for realizing the scene content. These complexities include generating online scene description that must be encoded and transmitted with the scene content to facilitate the decoding of this scene content on the other side of the network.
They also include providing elementary stream synchronization, multiplexing, and de-multiplexing mechanisms to facilitate data delivery over packet networks. Such complexities make special types of applications, such as broadcast applications, challenging and difficult to implement. This is due to the special features within broadcast as a way for information distribution including the redundancy mechanisms that facilitate broadcast listeners to listen to object-based presentation at any point during the presentation.

The MPEG4 systems standards specify technologies for a new generation of multimedia applications [2]. MPEG-4 was developed to permit real-time display and presentation of mixed media (audio, video, 3D, 2D, data) delivered over any TCP/IP network and medium: Internet, intranet, wireless, mobile, optical, MPEG-2, cable networks, etc. MPEG-4 will serve as the foundation for new media products and services such as delivery of high quality video and audio content via satellite to individual motor vehicles, PDAs, and virtually any display device without re-encoding. In addition to superior compression to MPEG-2, MPEG-4 also provides for object-oriented processing of the video clips, still images, text and computer graphics and user interaction that constitute the typical web experience but now contained in a multimedia stream.

Adoption of MPEG-4 is rapidly accelerating around the world. Companies such as Motorola, AT&T, Thomson, Toshiba, Philips, Cisco, Bertelsmann, and Sony all have announced plans to adopt MPEG-4 or are already implementing it. Yet, until the moment of writing the thesis, we believe there is no credible alternative yet proposed for broadcast operations using object-based presentations. This fact outrages the broadcast framework developed through this research work to have distinct value in providing delivery and
synchronization mechanisms for broadcasting MPEG4 audio-visual objects over packet networks.

1.3 Objectives and contributions

The main objective of this thesis is to build a working broadcast framework that facilitates the delivery of MPEG4 audio-visual objects over packet network. In that regard, the main contributions of the thesis are:

1. Developed new client and server elementary stream management layers using MPEG4 DMIF standards part 6 (ISO/IEC 14496-6) [3].
2. Developed new client and server service advertisement layers using session announcement protocol, recommended by IETF RFC 2974, and session description protocol, recommended by IETF RFC 2327.
3. Developed a server multi-service management layer to overcome the limitation of the MPEG4 reference software used during the implementation.
4. Developed a measurement-based algorithm for server bandwidth control to demonstrate the object-based effect on the MPEG4 media delivery.

The history of this framework started with the idea of building a delivery model for MPEG4 broadcast using ISO/IEC 14496-6, which defines a semantic framework for the Delivery Multimedia Integration Framework (DMIF). The standard covers three main technologies for communication, namely: interactive network technology, broadcast technology and the storage technology. Although the standard provides comprehensive definitions for elementary stream management, signalling, and synchronization in case of
interactive network technology, the specifications for the broadcast technology were still in the elementary stages. This developed the challenge of devising a new model for service and media delivery broadcast using a combined approach. This approach adopts the DMIF standard for MPEG4 elementary stream management and uses session announcement protocol for service and broadcast channels advertisement. The proposed broadcast framework also implements and accommodates the three main scenarios, namely: unicast, broadcast, and multicast, for services description and elementary streams content transport.

For realizing this framework, a client/server implementation has been developed for service and stream management. The implementation uses MPEG4 reference software called IM1 for elementary streams creation, decoding, and synchronization (see Appendix A). The implementation addresses some problems associated with the limitations of IM1 from the perspective of MPEG4 broadcast. These limitations are related to multiple service handling. To overcome these service-handling limitations, a new layer has been implemented to handle multiple services on the server side. This layer interfaces with IM1 to facilitate multiple-service handling and measures service network resources to provide service and channel bandwidth control through a simple, measurement-based, algorithm. This algorithm is used for manipulating data channels transport to satisfy the network bandwidth constraints.

1.4 Framework benefits

The framework adopts the push model for broadcast service advertisement. The main idea of this model is that clients do not have to connect to servers, and fetch information themselves. Instead, server pushes information to all the subscribed clients and clients start listening at their convenience. This indicates minimum information flow in the upstream
direction or from the client to the server. Such model is more compliant with the growth of interest for information broadcast using thin clients running on special devices (e.g. handheld devices, PDAs, cellular phones) or on special network platforms (e.g. Satellite, and wireless networks) and special types of asynchronous network connections like ADSL (Asynchronous direct subscriber line).

The revolutionary object-based realization for broadcast channels enhances the broadcast system over the traditional broadcast systems. This new concept enables maximum flexibility during scene interaction, encoding and transport. It also allows the scene content reusability and scalability to be able to match network and/or coding resource constraints.

The framework adopts the MPEG4 DMIF standard, which addresses the delivery issues in heterogeneous network environments and helps to develop applications that are network unaware through defining a semantic API for managing data delivery.

The framework also defines the transfer of service description information and elementary stream content through the three main transport scenarios (i.e. unicast, broadcast, multicast). This enables the broadcast channels to be scaled to reach huge number of clients simultaneously without affecting the server resource consumption.

1.5 Scope and outline

The thesis focuses on the delivery part of MPEG-4 media streams in broadcast applications. Chapter 2 builds the background about the system by providing an overview of the MPEG-4 standards and live media broadcast. This chapter discusses the main and specific
requirements for MPEG-4 systems, the MPEG-4 terminal architecture, and the major tools that the standards define for MPEG-4 applications regarding stream management, synchronization and media delivery. The chapter also illustrates the meaning of live media broadcast and discusses the main features and benefits that lead the broadcast delivery scenario to become one of the potential candidates for multimedia content distribution.

Chapter 3 covers the main architecture of the broadcast system. This chapter discusses the major parts and modules of the broadcast framework, including service advertisement and data delivery. It also covers some of the issues that the framework addresses within each part such as "client random access" and "data channels multiplexing".

Chapter 4 covers the client and server implementations and the major issues addressed in server implementation for realizing MPEG-4 media streams delivery. The chapter covers the techniques developed within the broadcast server implementation for data channels synchronization, and multiple service management. The chapter also discusses the bandwidth control algorithm developed to enhance the bandwidth utilization in multi-object presentations.

Performance analysis and experiments for verifying techniques and algorithms in chapter 4 are outlined in chapter 5. The chapter starts by discussing the effect of data channels synchronization on the media streams delivery. It then discusses the trade off investigation for data channels overhead and client start-up delay. The final section discusses the try-outs that verify the performance of the ServicePacketDrop and ObjectPacketDrop bandwidth control algorithms.
The thesis concludes in chapter 6 by highlighting the results drawn from building the broadcast framework and the potential directions for future work and implementation enhancements.
Chapter 2  MPEG-4 Broadcast Systems

The multimedia technology has drawn the industry to a new generation of information technology. Several standards have been emerged to accommodate the communication, and storage of multimedia information using wide variety of different delivery and storage techniques. Throughout the past decade, MPEG-4 standards has been emerging to provide multimedia stream syntax and a set of tools and interfaces for designers and builders to implement wide variety of MPEG-4 applications. The main theme of MPEG4 standards is to promote the presentation of multimedia content as a set of audio-visual objects that can be presented, manipulated, and transported independently. This concept has drawn much attention in different fields due to the advantages that can be gained through providing different levels of QoS for the separate audio-visual objects. Some of these advantages are related to the transport of the audio-visual objects and may include down-streaming (or sometimes dropping) some low-priority objects to lower the transfer rates for low bit rate network environments. Others are related to objects encoding and may include down-sampling, up-sampling, increasing or decreasing the error recovery overhead based on the object priority within the scene context. This ultimate flexibility makes MPEG 4 a new candidate for vast number of multimedia applications that handle the different delivery technologies including, local media, remote media access and live media broadcast. This chapter provides an overview of the MPEG4 standards highlighting the live media broadcast as one of the common scenarios for multimedia applications.
2.1 MPEG4 requirements

The MPEG-4 systems standard Error! Reference source not found. explains the MPEG4 systems requirements, which define the objectives that must be satisfied by any MPEG4 specifications. MPEG-4 Systems requirements may be categorized into two groups, namely: Traditional MPEG Systems, and Specific MPEG-4 Systems Requirements.

2.1.1 Traditional MPEG Systems requirements

The core requirements for the development of the systems specifications in MPEG-1 and MPEG-2 were to enable the transport of coded audio, video and user-defined private data, and to incorporate timing mechanisms to facilitate synchronous decoding and presentation of these data at the client side. These requirements also constitute a part of the fundamental requirements set for MPEG-4 Systems.

For data transportation, the audio-visual information is to be delivered in a streaming manner, suitable for live broadcast of such content. In other words, the audio-visual data is to be transmitted piece by piece, in order to match the delivery of the content to clients with limited network and terminal capabilities. This is in stark contrast to some of the existing scenarios, the World Wide Web, for example, wherein the audio-visual information is completely downloaded on the client terminal and then played back. It was thought that such scenarios would necessitate too much storage on the client terminals for applications foreseen by MPEG.
Synchronization of these streams is another issue where the different components of an audio-visual presentation are closely related in time. For most applications, audio samples with associated video frames have to be presented together to the user at precise instants in time. The MPEG representation needs to allow a precise definition of the notion of time so that data receives in a streaming manner can be processed and presented at the right instants in time, and be temporally synchronized with each other.

Once the audio-visual streams are created and the synchronization information has been added, a certain mechanisms have to be adapted to allow an application to consume the content. These include mechanisms for unambiguous location of the content, identification of the content type, description of the dependencies between content elements, access to the intellectual property information associated to the data.

All the previous issues for systems requirements are still relevant for MPEG4 and have led to the definition of some tools as well as enhancing some of the existing tools for preceding MPEG specifications. For example, Systems decoder model (SDM) is different from the ones dealt with in the traditional MPEG1 and MPEG2 decoder models. The main reason is the nature of the MPEG4 streams that may have a bursty data delivery schedule. Also, the broader range of supported bit rates for MPEG4 streams, compared with traditional MPEG standards, has led to the definition of a flexible tool to encode the synchronization information, the Synch Layer (SL). Moreover, due to the possibly unpredictable temporal behaviour of MPEG-4 data streams as well as the possibly large number of such streams in MPEG-4 applications, MPEG-4 Systems developed an efficient and simple multiplexing tool to enhance the transport of MPEG-4 data: the FlexMux (Flexible Multiplex) tool.
2.1.2 Specific MPEG-4 Systems requirements

The requirements in this set of systems requirements represent the ideas exclusive to MPEG-4 and are completely new in MPEG Systems. This is based on the main concept of MPEG4, which is coding of audio-visual objects. As per MPEG-4 terminology, an audio-visual object is the representation of a natural or synthetic object that has an audio and/or visual manifestation. Examples of audio-visual objects include a video sequence (perhaps with shape information), an audio track, an animated 3D face, speech synthesized from text, or a background consisting of a still image. The advantages of coding audio-visual objects can be summarized as follows:

1. It allows interaction with the content, which means, at the client side, users can be given access to manipulate or activate specific parts of the scene content.
2. It improves reusability and coding of the content. On the encoder side, authors can easily organize and manipulate individual components and reuse existing material. Moreover, each type of content can be coded using the most elective algorithms. This may be used to avoid the Artifacts due to joint coding of heterogeneous objects (e.g., graphics overlaid on natural video).
3. It allows content-based scalability. At various stages in the authoring, delivery, or consumption processes, content can be ignored or adapted to match bandwidth, diversity, or price requirements.

To be able to use these audio-visual objects in a presentation, additional information needs to be transmitted to the client terminals. The individual audio-visual objects are only a part of the presentation structure that an author wants delivered to the consumers. Indeed, for
the presentation at the client terminals, the coding of audio-visual objects needs to be augmented by:

1. The coding of information that describes the spatio-temporal relationships between the various audio-visual objects present in the presentation content. In MPEG-4 terminology, this information is referred to as the Scene description information.
2. The coding of information that describes how time-dependent objects in the scene description are linked to the streamed resources actually transporting the time-dependent information.

These considerations imply additional requirements for the overall architectural design. For example, in addition to the identification of the location of the streams, object description information may need to be attached to the media streams. This information may include the identification of streams in alternative formats, as well as a scalable stream hierarchy that is attached to the object or description of the object coding format. This object description information is conceptually different from the scene description information. It will therefore have different life cycles. For example, at some instant in time, object description information may change (like the intellectual property rights of a stream or the availability of new streams) while the scene description information remains the same. Similarly, the structure of the scene may change (like changing the positions of the objects), while the streaming resources remain the same. The consumer of the content may wish to obtain information about the content (e.g., the intellectual property attached to it or the maximum bit-rate needed to access it) before actually requesting the stream content. He/She
then only needs to receive information for the object description, not for the scene description.

2.2 MPEG4 Terminal Architecture

The overall architecture of an MPEG-4 terminal is depicted in Figure 2.1. The architecture has three main layers, namely: Compression Layer, Sync Layer, and Delivery Layer. The layering by the standard adds all the benefits of a modular design and ensures that each layer is only aware of the information that is meaningful to it.

Starting at the bottom of the figure, we first encounter the particular storage or transmission medium. This refers to the lower part of the delivery layer. The transport of the MPEG-4 data can occur on a variety of delivery mechanisms. This includes MPEG-2 Transport Streams, UDP (User Datagram Protocol) over IP (Internet Protocol), ATM (asynchronous transfer mode) AAL2 (ATM adaptation layer 2), or MPEG-4 (MP4) files. Most of the currently available transport layer systems provide native means for multiplexing information. However, there are a few instances where this is not the case, like in GSM (Global Systems for Mobile communication) data channels. In addition, the existing multiplexing mechanisms may not fit MPEG-4 needs in terms of low delay, or they may incur substantial overhead in handling the expected large number of streams associated with an MPEG-4 session. As a result, the FlexMux tool can optionally be used on top of the existing transport delivery layer. Regardless of the transport layer used and the use (or not) of the FlexMux option, the delivery layer provides to the MPEG-4 terminal a number of elementary streams. Note that not all of the streams have to be downstream (server to the client); in other words, it is possible to define elementary streams for the purpose of
conveying data back from the terminal to the transmitter or server. In order to isolate the design of MPEG-4 from the specifics of the various delivery systems, the concept of the DMIF (delivery multimedia integration framework) application interface (DAI) was defined. This interface defines the process of exchanging information between the terminal and the delivery layer in a conceptual way, using a number of primitives. It should be pointed out that this interface is non-normative; MPEG-4 terminal implementations do not need to expose such interface. The DAI defines procedures for initializing an MPEG-4 session and obtaining access to the various elementary streams that are contained in it. These streams can contain a number of different information: audio-visual object data, scene description information, control information in the form of object descriptors, as well as meta-information that describes the content or associates intellectual property rights to it. The delivery layer addresses the issues of local file access, broadcast media access, and peer-to-peer media access. Due to the exceeding demand in Internet multimedia, DMIF has sought to provide a flexible delivery platform for MPEG-4 and other multimedia technologies. The MPEG Delivery Multimedia Integration Framework (DMIF) enables each delivery technology to be used for its unique characteristics in a way transparent to the application developers.

Regardless of the type of data conveyed in each elementary stream, it is important that they use a common mechanism for conveying timing and framing information. The Sync Layer (SL) is defined for this purpose. It is a flexible and configurable packetization facility that allows the inclusion of timing, fragmentation, and continuity information on associated data packets. Such information is attached to data units that comprise complete presentation units, e.g., an entire video object plane (VOP) or an audio frame. These are called access units (AUs). An important feature of the SL is that it does not contain frame demarcation
information; in other words, the SL header contains no packet length indication. This is because it is assumed that the delivery layer that processes SL packets will already make such information available. Its exclusion from the SL thus eliminates duplication. The SL is the sole mechanism of implementing timing and synchronization mechanisms in MPEG-4. The fact that it is highly configurable allows the use of several different models. At one end of the spectrum, traditional clock recovery methods using clock references and time stamps can be used. It is also possible to use a rate-based approach (rather than using explicit timestamps, the known rate of the access units implicitly determines their time stamps). At the other end of the spectrum, it is possible to operate without any clock information where data is processed as soon as it arrives. This may be possible, for example, for a slide show presentation. The AU may also be larger than the SL packet, in which case it will be fragmented across multiple SL packets. At the receiving terminal, the AUs are reassembled and delivered to the Compression Layer. At the time of this writing, there are ongoing discussions to place numerous short AUs (e.g., low-bit-rate audio) in a single SL packet to increase efficiency and reduce overhead [12].
Figure 2.1 MPEG4 systems architecture

The SL produces time-based elementary streams in a form of AUs. Each elementary stream is sent to its corresponding decoder in the compression layer for decoding the composition information (e.g., a decoded video object plane). In order for the receiver to know what type of information is contained in each stream, control information in the form of
object descriptors is used. These descriptors associate sets of elementary streams to one audio or visual object, define a scene description stream, or even point to an object descriptor stream. These descriptors, in other words, are the way with which a terminal can identify the content being delivered to it. Unless a stream is described in at least one object descriptor, it is impossible for the terminal to make use of it. At least one of the streams must be the scene description information associated with the content. The scene description information defines the spatial and temporal position of the various objects, their dynamic behaviour, as well as any interactivity features made available to the user. As mentioned above, the audio-visual object data is actually carried in its own elementary streams. The scene description contains pointers to object descriptors when it refers to a particular audio-visual object. The scene description is tree-structured and is heavily based on VRML (Virtual Reality Modeling Language) structure. A key feature of the scene description is that, since it is carried in its own elementary stream(s), it can contain full timing information. This implies that the scene can be dynamically updated over time, a feature that provides considerable power for content creators. In fact, the scene description tools provided by MPEG-4 also provide a special lightweight mechanism to modify parts of the scene description in order to effect animation. This is accomplished by coding, in a separate stream, only the parameters that need to be updated. The system's compositor uses the scene description information, together with decoded audio-visual object data, in order to render the final scene that is presented to the user. It is important to note that the MPEG-4 Systems architecture does not define how information is to be rendered. In other words, the Systems part of the MPEG-4 standard does not detail mechanisms through which the values of the pixels to be displayed or audio samples to be played back can be uniquely determined. This is an unfortunate side effect of providing synthetic content representation tools. Indeed, in the general case, it is not possible to define
rendering without venturing into issues of terminal implementation. Although this makes compliance testing much more difficult (requiring subjective evaluation), it allows the inclusion of a very rich set of synthetic content representation tools. In some cases however, like in the Audio part of the MPEG-4 standard, composition can be and is fully defined. The scene description tools provide mechanisms to capture user or system events. In particular, they allow the association of events to user operations on desired objects, that can, in turn, modify the behaviour of the stream. Event processing is the core mechanism with which application functionality and differentiation can be provided. In order to provide flexibility in this respect, MPEG-4 allows the use of ECMA Script (also known as JavaScript) scripts within the scene description. Use of scripting tools is essential in order to access state information and implement sophisticated interactive applications. It is important to point out that, in addition to the new functionalities that MPEG-4 makes available to content consumers, it provides tremendous advantages to content creators as well. The use of an object-based structure, where composition is performed at the receiver, considerably simplifies the content creation process. Starting from a set of coded audio-visual objects, it is very easy to define a scene description that combines these objects in a meaningful presentation. A similar approach is essentially used in HTML (Hyper Text Mark-up Language) and Web browsers, thus allowing even non-expert users to easily create their own content. The fact that the content's structure survives the process of coding and distribution, also allows for its reuse. For example, content filtering and/or searching applications can be easily implemented using ancillary information carried in object descriptors (or its own elementary streams). Also, users themselves can easily extract individual objects, assuming that the intellectual property information allows them to do so.
2.3 MPEG4 systems tools

This section describes the main aspects to MPEG4 standard tools. These tools are used for elementary streams realization, scene composition, elementary streams timing and synchronization, as well as the transport or MPEG4 elementary streams over variety of different networks environments. It's important to notice throughout the section the motivation of implementing these tools which derives from the object based model of MPEG4 and the diverse transportation frameworks which could be used in conjunction with each other to compose the MPEG4 presentation.

2.3.1 System Management

The object description framework provides the glue between the scene description and the streaming resources - the elementary streams - of an MPEG-4 presentation, as indicated in [4]. Unique identifiers are used in the scene description to point to the object descriptor, the core element of the object description framework. The object descriptor is a container structure that encapsulates all of the setup and association information for a set of elementary streams. A set of sub-descriptors, contained in the object descriptor, describe the individual elementary streams, including the configuration information for the stream decoder as well as the flexible sync layer syntax for this stream. Each object descriptor, in turn, groups a set of streams that are seen as a single entity from the perspective of the scene description. Object descriptors are transported in dedicated elementary streams, called object descriptor streams that make it possible to associate timing information to a set of object descriptors. With the appropriate wrapper structures, called OD commands (object descriptor commands), around each object descriptor, it is possible to update and remove each object.
descriptor in a dynamic and timely manner. The existence or the absence of descriptors determines the availability of the associated elementary streams to the MPEG-4 terminal. The initial object descriptor, a special case of the object descriptor, is a key element necessary for accessing the MPEG-4 content. It conveys content complexity information in addition to the regular elements of an object descriptor. As depicted in Figure 2.2, the initial object descriptor usually contains at least two elementary stream descriptors. One of the descriptors must point to a scene description stream while the others may point to an object descriptor stream. This object descriptor stream transports the object descriptors for the elementary streams that are referred to by some of the components in the scene description. Initial object descriptors may themselves be transported in object descriptor streams since they allow content to be hierarchically nested, but may as well be conveyed by other means, serving as starting pointers to MPEG-4 content. In addition to providing essential information about the relation between the scene description and the elementary streams, the object description framework provides mechanisms to describe hierarchical relations between streams, reflecting scalable encoding of the content and means to indicate multiple alternate representations of content. Furthermore, textual descriptors about content items, called object content information (OCI), and descriptors for the intellectual property rights management and protection (IPMP) have been defined. The latter allow conditional access or other content control mechanisms to be associated to a particular content item. These mechanisms may be different on a stream-by-stream basis and possibly even a multiplicity of such mechanisms could co-exist. A single MPEG-4 presentation, or program, may consist of a large number of elementary streams with a multiplicity of data types. The object description framework has been separated from the scene description to account for this fact and the related consequence that service providers may possibly wish to relocate streams in a simple way. Such relocation
may require changes in the object descriptors; however, it will not affect the scene
description. Therefore, object descriptors improve content manageability.

Figure 2.2  Object descriptors linking scene description to elementary
streams

2.3.2 Object Presentation (composition): BIFS

MPEG-4 specifies a binary format for scenes (BIFS) that is used to describe scene
composition information: the spatial and temporal locations of objects in scenes, along with
their attributes and behaviours. Elements of the scene and the relationships between them
form the scene graph that must be coded for transmission. The fundamental scene graph
elements are the “nodes” that describe audio-visual primitives and their attributes, along with
the structure of the scene graph itself. BIFS draws heavily on this and other concepts
employed by VRML that is designed as a file format for describing 3D models and scenes ("worlds" in VRML terminology), VRML lacks some important features that are required for the types of multimedia applications targeted by MPEG-4. In particular, the support for natural video and audio are basic (ex: streaming of audio or video objects are not supported) and the timing model is loosely specified, implying that synchronization in a scene consisting of multiple media types cannot be guaranteed. Furthermore, VRML worlds are often very large (ex: there is neither compression nor animations streaming). Animations lasting around 30 s typically consume several megabytes of disk space. The strength of VRML is its scene graph description capabilities and this strength has been the basis upon which MPEG-4 scene description has been built. BIFS includes support for almost all of the nodes in the VRML specifications. In fact, BIFS is essentially a superset of VRML, although there are some exceptions. BIFS does not yet support the PROTO and EXTERNPROTO nodes, nor does it support the use of Java language in the Script nodes (BIFS only supports ECMA Script). BIFS does, however, expand significantly on VRML's capabilities in ways that allow a much broader range of applications to be supported. Note that a fundamental difference between the two is that BIFS is a binary format, whereas VRML is a textual format. So, although it is possible to design scenes that are compatible with both BIFS and VRML, transcoding of the representation formats are required.

BIFS describes an efficient binary representation of the scene graph information. This information can be transferred in a streaming manner in a way that enables transferring initial scene followed by time-stamped modifications to the scene. This information may also describe the continuous animation changes including facial animation (i.e. BIFS accommodates the transfer of the animated face properties at the scene level) in a form of
numerical values of the components in the scene. Finally, BIFS provides enhanced audio support in the notion of “audio scene graph” where the audio sources, including streaming ones, can be mixed. Audio contents within MPEG-4 scene can even be processed and transformed with special procedural code to produce various sounds effects.

2.3.3 Timing and synchronization

Timing of streams is expressed in terms of decoding and composition time of individual access units within the stream. Access units are the smallest sets of data to which individual presentation time stamps can be assigned (e.g., a video object plane). The decoding time stamp indicates the point in time at which an access unit is removed from the decoding buffer, instantaneously decoded, and moved to the composition memory. The composition time stamp allows the separation of decoding and composition times, to be used for example in the case of bi-directional prediction in visual streams. This idealized model allows the encoding side to monitor the space available in the decoding side's buffers, thus helping it, for example, to schedule ahead-of-time delivery of data. Of course, a resource management for the memory for decoded data would be desirable as well. However, it was acknowledged that this issue is strongly linked with memory use for the composition process itself, which is considered outside the scope of MPEG-4. Therefore, management of composition buffers is not part of the model. Time stamps are readings of an object time base (OTB) that is valid for an individual stream or a set of elementary streams. At least all the streams belonging to one audio-visual object have to follow the same OTB. Since the OTB, in general, is not a universal clock, object clock reference time stamps can be conveyed periodically with an elementary stream to make it known to the receiver. This is, in fact, done
by the synchronization layer (SL). The sync layer provides the syntactic elements to encode the partitioning of elementary streams into access units and to attach both decoding and composition time stamps as well as object clock references to a stream. The resulting stream is called an SL-packetized stream. This syntax provides a uniform shell around elementary streams, providing the information that needs to be shared between the compression layer and the delivery layer in order to guarantee timely delivery of each access unit of an elementary stream. Different from its predecessor in MPEG-2, the packetized elementary stream (PES), the sync layer does not constitute a self-contained stream, but rather a packet-based interface to the delivery layer. This takes into account the properties of typical delivery layers like IP, H.223 or MPEG-2 itself, into which SL-packetized streams are supposed to be mapped. There is no need to encode either unique start codes or the length of an SL packet within the packet, since synchronization and length encoding are already provided by the mentioned delivery layer protocols. Furthermore, MPEG-4 has to operate both at very low and rather high bit-rates. This has led to a flexible design of the sync layer elements, making it possible to encode time stamps of configurable size and resolution, as required in a specific content or application scenario. The flexibility is made possible by means of a descriptor that is conveyed as part of the elementary stream descriptor that summarizes the properties of each (SL-packetized) elementary stream.

2.3.4 MPEG-4 delivery

Delivery of MPEG-4 content is a task that is supposed to be dealt with outside the MPEG-4 systems specification. The Delivery Multimedia Integration Framework has a number of objectives including hiding the delivery technology details from the DMIF User,
managing the real time, QoS sensitive channels, as well as, ensuring the interoperability between end-systems.

All access to delivery layer functionality is conceptually done only through a semantic interface, called the DMIF application interface (DAI). It is specified in Part 6 of MPEG-4, Delivery multimedia integration framework (DMIF) [3]. In practical terms this means that the specification of control and data mapping to underlying transport protocols or storage architectures is to be done jointly with the respective organization that manages the specification of the particular delivery layer. For example, for the case of MPEG-4 transport over IP, development work is done jointly with the Internet Engineering Task Force (IETF).

The integration framework of DMIF covers three major technologies, interactive network technology, broadcast technology and the storage technology. Figure 2.3 represents the DMIF communication architecture. As mentioned before, the main objective of the DMIF layer is hiding the communications method from the application. So, the implementation of DMIF takes care of the delivery technology details presenting a common interface to the application.

Figure 2.3 depicts the above concept. An application accesses data through the DMIF-Application Interface, irrespective whether such data comes from a broadcast source, from local storage or from a remote server. A Delivery layer implementation allows the concurrent presence of more than one DMIF Instance. When requesting a specific service the Application supplies a URL that allows the Delivery layer to determine the appropriate DMIF Instance to activate. The DMIF Instances will then translate the Originating Application
requests into specific actions to be taken with respect to the broadcast network or the local file system, or translate it into messages to be delivered to the Target Application, taking care of the peculiarities of the involved delivery technology. Similarly, data entering the terminal (from remote servers, broadcast networks or local files) is uniformly delivered to the Originating Application through the DAI. The DAI allows the DMIF User to group elementary streams into services and to specify the QoS requirements for the desired elementary streams.

DMIF separates the common features that should be implemented in each DMIF Instance from the details of the delivery technology. In the case of interactive networks DMIF specifies a logical interface (the DMIF-Network Interface – DNI) between the module implementing this common features and the network specific modules. The DNI specifies the information flow that should occur between peers of such hypothetical module; the network specific modules specify the exact mapping of the DNI primitives into signalling messages. DNI primitives need to be adapted to the native network signalling mechanisms in order to provide access to the network resources. DNI primitives represent one of the possible solutions to ensure the DAI functionality when interacting with a remote peer. A straightforward mapping of DNI primitives into signalling messages is defined, and named DMIF Default Signalling Protocol (DDSP). The DMIF Default Signalling Protocol is a session level protocol for the management of multimedia streaming over generic delivery technologies. The protocol comprises primitives to set-up and tear down sessions as well as individual data channels.
DNI primitives, however, are not valid for the Broadcast and Local Storage scenarios. We will discuss the broadcast scenario in details in the following chapters and describe another methodology for negotiating service-specific information between peers. It's only important at this moment to know that, in the case of the Broadcast and Local Storage scenarios the model is simpler, and no internal interface has been identified. Conceptually, such DMIF Instances include the features of a target DMIF peer, as well as those of the Target Application. All the control messages that are exchanged between peer applications in an interactive scenario are in this case terminated in the DMIF Instance. Stream control commands like PLAY/PAUSE/MUTE are examples where these commands should be created, issued, and executed on both peers' DMIF instances and not signalled on the other peer. Figure 2.4 shows the possible delivery technologies that DMIF standards address.
2.4 Live Media Broadcast

Broadcast distribution is a recent method for transmitting live broadcast data from an originating server (a transmitter) to a receiving terminal (a receiver). The transmitter is configured to distribute content from the live source to one or more receivers, with almost no upper limit on the number of receivers. The receiver is configured to listen for live broadcasts that originate from the transmitter. Live broadcast distribution can be initiated through data channels either in a push manner, where the transmitter initiates the packet flow to the receiver, or in a pull manner, where a client request triggers the transmitter to begin sending data packets.
2.4.1 Features and Benefits

- **Scalability**

  Live broadcast distribution technology has proven to be the most scalable means currently available for delivering streaming media to vast number of audiences. This is due to the fact that the server initiates and feeds one service instance to all the listening clients. This is different from the remote retrieval scenario, where the server must initiate one media instance for each client request. This promotes the broadcast scenario over the remote retrieval scenario in terms of scalability and bandwidth utilization.

- **Optimization for Connectionless Transmissions**

  Live broadcast distribution is optimized to write streaming media packets over a network in a unidirectional manner that does not require acknowledgment of arrival from the packet receiver. By default, broadcast distribution requires only a single one-way network path from the transmitter to the receiver using one of the common transport scenarios (i.e. unicast, broadcast, multicast). If connectivity between the transmitter and receiver is interrupted and subsequently restored, the live distribution will continue without needing to be restarted and without any other action required on the part of either the transmitting or receiving server. The necessary session description, stream data, and error correction data flow across the network to the receiver, so there is no need for an upstream-channel. This saves the delay of establishing a connection with the server and waiting for a feedback from the client for each packet.
• Using multiple transport scenarios

Using broadcast and multicast as transport scenarios saves the server time to send stream content as multiple packet-flows to each receiver individually incurring the network delay for each network send. By using broadcast or multicast, server sends one packet-flow over the network and receivers either subscribe to the packet flow in case of multicast or listen to broadcast packets over the network. This is depicted in Figure 2.5. For more information on how to send packet-flows using broadcast and multicast, see Appendix B.

![Diagram of live broadcast distribution using multiple transport scenarios](image)

**Figure 2.5** Live broadcast distribution using multiple transport scenarios
Chapter 3 MPEG-4 Broadcast System architecture

In this chapter we explain the architecture of our broadcast framework highlighting the issues that are distinct to the broadcast as a scenario for realizing information streams and the rationale behind the system design to accommodate each of these issues. Although our experiments and the system implementation, explained in Chapter 5, are based on the IP network environments, the issues discussed in the following sections will still be valid for other network technologies (e.g. ATM, wireless networks, etc.).

Figure 3.1 Broadcast System architecture

MPEG-4 broadcast system architecture is based on the push model described in [1]. The system architecture is composed of two main layers: the broadcast service advertisement layer, and the data delivery layer. Both layers could adopt unicast, broadcast or multicast as a way of communication amongst the network terminals. In case of broadcast/multicast, the
server sends the layer specific information to the broadcast network and may not know who will be listening to this information. In case of broadcast, the information is sent over the network broadcast domain (i.e. to a specific network port) whereas in multicast, the information will be sent to an IP multicast address which will resolve to a list of clients' addresses that joined the multicast session and only these clients will be able to listen to the incoming information. The system architecture is depicted in Figure 3.1. The following sections will highlight the main issues and methodology related to each layer.

3.1 Broadcast Service Advertisement

The Broadcast Service advertisement layer is based on the publisher-subscriber framework, which is part of the “push” model described in [1]. The essence of this concept is that users should sign up for applications that send the information directly to them, without having to fetch the information themselves. The Broadcast Service Advertisement layer is responsible for maintaining and manipulating the available services. This could be done through the use of two main concepts in conjunction to each other.

- Advertisement: A server (publisher) provides channel information to clients.
- Subscription: clients (subscribers) receive and consume channel information to start data delivery.

These concepts are depicted in Figure 3.2 and explained more in the following sections.
3.1.1 Service advertisement

The service advertisement is done when the server broadcasts/multicasts the information about the available broadcast channels (services). This information includes service description fields and media description fields. The service description fields describe the broadcast channel, and provide the machine address where clients can send the request to subscribe to this channel if needed. Media description fields specify the transport protocol and network ports used for each media stream within the service. There are number of protocols that may be used to encode the service information.

Session description protocol (SDP) [9] is one of the text-based protocols that may be used to convey the information about the services including UDP ports for data streams and the time interval when these media streams are active. Table 3.1 lists the service channel description fields and media description fields used in session description protocol. The detailed description of all the fields and their respective sub-fields is described in [9]. Although SDP is intended to be general purpose so that it can be used for a wide range of network environments, it does not incorporate any transport or control mechanism, and
therefore must be used in conjunction with another protocol for conveying channel description information such as SIP, RTSP, or SAP.

<table>
<thead>
<tr>
<th>Service description</th>
<th>Multiple flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>V Protocol version</td>
<td></td>
</tr>
<tr>
<td>O Owner/creator and session identifier</td>
<td></td>
</tr>
<tr>
<td>S Session name</td>
<td></td>
</tr>
<tr>
<td>I Session information</td>
<td></td>
</tr>
<tr>
<td>U URI of description</td>
<td></td>
</tr>
<tr>
<td>E Email address</td>
<td></td>
</tr>
<tr>
<td>P Phone number</td>
<td></td>
</tr>
<tr>
<td>C Connection information - not required if included in all media</td>
<td></td>
</tr>
<tr>
<td>B Bandwidth information</td>
<td></td>
</tr>
<tr>
<td>Z Time zone adjustments</td>
<td></td>
</tr>
<tr>
<td>K Encryption key</td>
<td></td>
</tr>
<tr>
<td>A Session attribute lines</td>
<td>Zero or more</td>
</tr>
<tr>
<td>T Time the session is active</td>
<td></td>
</tr>
<tr>
<td>R Repeat times</td>
<td>Zero or more</td>
</tr>
<tr>
<td>Media description</td>
<td>Zero or more</td>
</tr>
<tr>
<td>M Media name and transport address</td>
<td></td>
</tr>
<tr>
<td>I Media title</td>
<td></td>
</tr>
<tr>
<td>C Connection information - optional if included at session-level</td>
<td></td>
</tr>
<tr>
<td>B Bandwidth information</td>
<td></td>
</tr>
<tr>
<td>K Encryption key</td>
<td></td>
</tr>
<tr>
<td>A Zero or more media attribute lines</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.1  Session description protocol fields**
Unlike most of the protocols for service advertisement, which require clients to send feedback to the server during the service setup, Session Announcement Protocol (SAP) [8] does not require two-way communication for advertising broadcast services between server and respective clients. Therefore, it is the protocol proposed by our architecture to broadcast/multicast the service channel information to all clients. The SAP packet contains the SDP information as well as a SAP header for manipulating the service channel information. The SAP header contains address type, message type fields, as well as a unique message identifier for each service channel announcement. The SAP header also contains authentication, compression, and encryption fields for each service channel as shown in Figure 3.3. These fields indicate whether the server uses authentication, compression, or encryption during the SAP header encoding. Session announcement protocol standard [8] explains the SAP header fields in details.

The authentication may be used to verify that manipulating the service is permitted and/or to authenticate the identity of the service creator. SAP is not tied to any single authentication mechanism. The structure of the format specific authentication sub-header including the PGP and CMS authentication mechanisms is described in [13].

Compression and/or encryption may also be performed on the SAP header before authentication is added. While compression saves network bandwidth, it increases the delay on both the announcer and the listener sides. It also adds another source of error to the SAP packet if the client cannot uncompress the payload due to a detected error. Therefore, compression is only recommended in some situations when the network bandwidth is critical and the rate of announcements is relatively high.
### Figure 3.3 SAP Header fields

Unlike compression, which may be used to save bandwidth, encryption may become a waste of bandwidth if many of the receivers who are listening to the announcement do not have the encryption key. For this reason, the use of encrypted SAP announcement is not recommended except for some special cases where encryption may be mandatory to prevent malicious access to the broadcast services. Besides, if the announcement is made through a proxy, there may be no way for the proxy to discover that the announcement is superseded, and hence it may continue passing the old announcement and increase the waste of bandwidth.

SAP packets are transported using best-effort UDP/IP protocol stack in low bit rate and do not require any additional reliability in terms of packet loss detection. During the
announcement process, server broadcasts, or multicasts to a special IP multicast address, the service information repeatedly. The time period between repetitions is chosen such that the total bandwidth used by all announcements on a single SAP group remains below a reconfigured limit (i.e. this limit is recommended to be 4000 bits per second). Hence, the actual transmission time for every new announcement is derived from the base interval as follows:

- Given a configured bandwidth limit \( L \) in bits/sec for \( N \) announcements and an announcement size \( S_i \), the announcement interval \( I \) in seconds is

\[
I = \frac{\text{MAX} (300 ; (8 \times N \times S_i))}{L} \quad \text{EQ 3-1}
\]

- An offset \( O \) is then calculated based on the base interval

\[
O = \text{RAND} (2I / 3) - \frac{I}{3} \quad \text{EQ 3-2}
\]

- The next transmission time for the announcement is derived as

\[
t_n = t_p + I + O \quad \text{EQ 3-3}
\]

Where \( t_p \) is the time of sending the previous announcement. All SAP listeners dispatched by the clients receive the SAP packets and categorize them based on the SAP header information (i.e. New announcement, remove announcement). As a result of transferring SAP messages in the low bit rate without error detection, clients may suffer from long delays during start-up until they receive all the information about the available broadcast services. Such a problem may be resolved using SAP proxy caches, which maintain an up-to-date list of all the broadcast services. The client may then contact this proxy cache during start-up to get the service channel information.
3.1.2 Service Subscription

The subscription process is only valid if the server is unicasting the media streams. During subscription, clients send a timed-out request to subscribe or unsubscribe from a broadcast channel sending the client host information, client credentials, and possibly the client capabilities as shown in Figure 3.4.

<table>
<thead>
<tr>
<th>Message type</th>
<th>Client host information</th>
<th>Message ID</th>
<th>Client Credentials</th>
<th>Client Requirements</th>
</tr>
</thead>
</table>

**Figure 3.4   Client subscription PDU**

The subscription PDU includes a 1-bit message type field which determines if the message is a SUBSCRIBE or UNSUBSCRIBE request. The packet also includes the size and the payload for the client host information, which follows the format of the owner information field in the SDP information explained in [9]. 16-bit Message ID specifies particular service to subscribe to. Client credentials are used to verify clients against any malicious access to the broadcast channels. Clients’ requirements may also be used so that the server may use it to customize the stream QoS parameters to accommodate diverse clients requirements.

Server may also delegate an intermediate proxy server to be responsible for accommodating the clients’ requirements. The proxy server then receives the subscription
requests, classifies clients into groups, and reformats the channel contents to match each group. This may happen by dropping some of the scene contents for each group. The idea of using grouping as criteria for routing the channel content may very well be efficient in case of using IP multicast [13][14]. This scheme is depicted in Figure 3.5.

![Figure 3.5 Clients grouping for IP multicast](image)

In case of errors, invalid subscription requests may cause long delays at service start-up. Therefore, the subscription request is sent to the server through a bi-directional error-free transport channel (e.g. TCP, or UDP with error recovery). If the client receives a positive acknowledgment from the subscription server, it may then start the service setup by invoking the delivery layer to start receiving the media streams. Based on the media stream information in the SDP payload, the client initializes the required delivery layer instances required to accommodate the received media streams. As shown in Figure 3.1, the broadcast service advertisement layer invokes the DMIF layer through the DMIF application interface.
3.2 Service Data delivery (DMIF)

As mentioned previously, the delivery layer of our MPEG-4 Streaming System architecture conforms to the recommendations made by the MPEG-4/DMIF standard. However, the broadcast service advertisement discussed in section Broadcast service advertisement replaces the signalling part of the DMIF standard and therefore only data delivery part (data plane) of the DMIF layer is explained in this section.

3.2.1 Streams delivery

To start media streams delivery, server reads the Initial Object Descriptor (IOD) data from the elementary stream provider (see Figure 3.6). Based on the network environment, client may ask server to send IOD or to wait for the server to broadcast the IOD repeatedly as explained in the next paragraph. Using this information, clients can locally create data channels required to receive the media streams described in the IOD (i.e. Scene descriptor(s)). Once data channels are created, client receives the scene descriptor (SD), which describes the relationship between various objects in the presentation, as well as the media streams descriptors for all the audio-visual objects. Based on the (IOD), (SD), and streams object descriptors, client recursively creates the data channels required to carry the various streams that correspond to these objects and starts receiving the media streams through these data channels. In this scenario, clients are passive elements receiving media streams and control messages sent by the server. A network/medium access sub-layer, (i.e. DMIF Medium Access Layer) is responsible for providing a transparent interface for Network connectivity as will be discussed in the next chapter.
Figure 3.6  DMIF data channels

There is a "bandwidth versus delay" trade off for sending the initial object descriptor from the server to all clients. Some clients are very susceptible to delay (e.g. satellite receivers, some wireless receivers, etc.). In this case, Server does not wait for clients’ requests to send the IOD. Instead, server broadcasts (or multicasts) service-attach messages periodically sending the IOD to all the listening clients (i.e. clients subscribed through the service management module). Some other network environments may not have this delay issue (e.g. LANs, WANs, etc.). In this type of networks, client sends a request to the server to start listening to the service channel and then server sends the IOD. In this case, clients will incur the delay of waiting for the server to send the IOD before starting listening to the service channel content.
3.2.2 Client random access

Although, the repetition of sending service descriptors information imposes a bandwidth overhead during stream packetization, it is mandatory for stream broadcast to allow clients to listen to the channel content at any time during the presentation. This concept is called channel random access. To enable channel random access, IOD is not the only information that has to be sent repeatedly during service presentation. Each elementary stream has a descriptor that holds the configuration information relevant to this elementary stream. Therefore, all these descriptors have to be sent repeatedly to enable random access. Service random access is depicted in Figure 3.7. The figure shows that the server sends the IOD, Scene descriptors, and object descriptors repeatedly, each in their relevant data channels. Clients can access this information at any time and use it to access the data channel content for the audio-visual elementary streams (sync layer packets).

![Figure 3.7 MPEG-4 DMIF broadcast model for client random access](image)
3.2.3 Calculating Data Channels overhead

Data channels whose content must be repeated during data delivery to allow clients to randomly access the stream content and any time instance cause overhead in the data channels. The main overheads that have to be contained in any service are IOD and ODs channel overheads. To calculate these overheads, we assume that the server broadcasts N services. Each service has an average bit rate of $R_i$ bits/sec for stream playback. We also assume that the maximum delay tolerance that the client can incur to get the IOD or the ODs is $T_m$ since the client’s service starts. So, the client must receive the required object descriptor packets within the $T_m$ time period.

The IOD channel overhead ($V_I$) may be simply calculated as follows:

$$V_I = \frac{N \sum L_i}{T_m \sum R_i}$$  \hspace{1cm} \text{EQ 3-4}

Where $L_i$ is the length in bits of the IOD data. For example, assume 10 service channels running with 50kb/sec data rate each; IOD data length of 1000 bits, and maximum delay time of 1 second, $V_I$ is almost 2%. For 500Kb/sec channels, $V_I$ is 0.2%.

The ODs channel overhead ($V_o$) may also be calculated as follows

$$V_o = \frac{N (m L_m + n L_n)}{T_m \sum R_i}$$  \hspace{1cm} \text{EQ 3-5}
$L_s$ is the scene descriptor length in bits, $m$ number of scene descriptors for the service channel, $L_o$ is the object descriptor length in bits, and $n$ is the number of object descriptor for the service channel.

For the same example explained above, assume each broadcast channel has 1 scene descriptor and 2 object descriptors for audio and video streams and average length of 1000 bits for each. Then, the total overhead is 5.9 % and 0.59% respectively.

Since $\Sigma R_i/N$ is actually the average bit rate $R_a$, we can eliminate the dependency on the number of services and rewrite both overheads as follows:

$$V_i = \frac{L_s}{R_a T_m}$$  \hspace{1cm} \text{EQ 3-6}

$$V_o = \frac{(m L_s + n L_o)}{R_a T_m}$$  \hspace{1cm} \text{EQ 3-7}

Both equations show that the overhead caused by IOD, SDs, and ODs does not depend on the number of broadcast services. Instead, it depends only on the average bit rate as well as the maximum delay tolerance for service channel startup.

**3.2.4 Data Channels multiplexing**

MPEG-4 standards promote the concept of conveying elementary streams on individual data channels. This separation enables the use of different QoS parameters for each group of data channels that share the same QoS profile. The QoS information is sent as part
of the Object Descriptor information [3] for each individual elementary stream. This QoS information may include encode/decode QoS parameters (e.g. Resolution, Quantization loss, etc.), and network QoS parameters (e.g. packet delay, packet loss, peak rate, average rate, etc.). Therefore, for the purpose of streams transmission, a multiplexing scheme may be adapted to classify elementary streams into groups that share the same network QoS parameters and then send each group of elementary streams into one network channel through the appropriate transport protocol (e.g. UDP).

As mentioned before, during media streams delivery, both server and respective clients create and control data channels locally. Therefore, there is no way of synchronizing data channel index between server and clients. In other words, the contents of the same data channel number might have different media contents on both server and client sides. This has to be taken into consideration while designing the multiplexing scheme for the delivery layer. The next section will explain how this issue affected our packetization technique for media streams delivery.

3.2.5 Transport packet's Header Format

The transport packet's header will be appended to the SL-Packets before sending across the network medium. The proposed header format for transport packets has to take into consideration some conceptual issues, which are affected by the nature of the broadcast environment over variety of network environments. These issues include:
• For one-way communication: clients should be able to distinguish the command (control) packets (i.e. IOD, SD, ODs) from the data packets.

• Packet repetition: clients should be able to distinguish new packets from old ones.

• Asynchronous Data channels multiplexing: data channels control and creation happen locally on both client and server.

Regarding these issues, the proposed header format is depicted in Figure 3.8.

![Transport Packet Header Format](image)

**Figure 3.8** Transport Packet Header Format

**Reserved**: 5-bit reserved field, set to 0.

**Packet Type**: 3 bit field, has the following types

- IODPacket: 0
- DataPacket: 1
- StopChannel: 2
- PauseChannel: 3

**ES_ID**: 16-bit multiplexing field that holds the elementary stream ID

**Packet ID**: 32-bit packet ID, identifier for the packet

**Checksum**: 16-bit header check sum
**Payload:** Payload field that holds the channel content

The reason for having a separate packet type for IOD packet is that IOD packet information is received by clients during the service attach phase. Whereas, data packets including ODs and SDs are received within their own relevant data channels. So, for IOD, there is a separate packet type (IODPacket) and for all data channels content we have one packet type (DataPacket). The stop and pause channel packet types, may be used to save resources on the clients' side by sending commands to clients to stop channel and remove channel resource or to pause receiving data on this channel.

For channel multiplexing, the ES_ID is used to distinguish which data channel holds which elementary stream. The reason for using ES_ID for channel multiplexing and not the channel number is the asynchronous nature of the channel multiplexing that our broadcast system has. In case of broadcast scenario, data channel manipulation (i.e. adding and removing data channels) happens locally on both client and server sides and there is no way of synchronizing the channel number between the server and client. In other words, same elementary stream may be handled by different channels on the server and the client.

Also, to enable channel random access by repeating some channel contents, clients should be able to recognize new packets from repeated old ones. This may happen if the server sends new SD information or OD while the old SD or OD is being repeated. In this situation, server uses different packet ID for each new packet that will be repeated. Clients will then capture the change and send the new packet to the higher layers for decoding.
In conclusion, this chapter has demonstrated the basic means for accommodating MPEG-4 broadcast applications including channels multiplexing and data delivery. To support client random access feature, the server must repeatedly send the contents of the object descriptor channels to allow clients to access the service presentation at any time. The overhead caused by repeating the object descriptor information is insignificant especially for high demand services. Special technique has to be adopted for data channels multiplexing due to asynchronous data channels manipulation (i.e. data channels are created and controlled locally at the server and the clients).
Chapter 4 MPEG-4 Broadcast System Design and Implementation

The start point of our broadcast system implementation is the IM1 reference software (see Appendix A). This reference software was used during system implementation and testing. The following are the main steps that we pursued to build our Broadcast system:

1. Built client and server service advertisement layer for service information announcement and subscription.
2. Built client and server DMIF delivery layer instances for Broadcast scenario.
3. Built wrapper layer around the server's system layer to accommodate multiple services (multiple-service layer).
4. Optimized the server load by manipulating the transfer of elementary streams using a measurement-based algorithm for streams bandwidth control.

This chapter explains the broadcast system functional design and implementation highlighting the main client and server modules and how they collaborate to realize MPEG-4 media streams broadcast. The chapter also discusses the challenges encountered during the system implementation including data channels synchronization and multiple service handling. The chapter also discusses the 2 proposed algorithms for bandwidth control, namely ServicePacketDrop and ObjectPacketDrop.

Both client and server implementations are based on the object-oriented modular design techniques. The resulting software incorporates the main modules required for
Broadcast Service Advertisement and DMIF Data Delivery layers. The modular design empowers scalability, interoperability, and enables the integration of the broadcast technology with other technologies (e.g. remote retrieval and local storage media).

**Figure 4.1 Module Diagram of Broadcast service advertisement**

Both implementations for client and server broadcast service advertisement are based on the service advertisement module diagram depicted in Figure 4.1. The delivery layers are based on the terminal architecture, for data delivery, depicted in Figure 4.2. However, there are differentiating issues for the client and server that will be discussed on their corresponding sections. The figure shows the main modules for delivery layer, the main objects inside each module and the interface through which each module collaborates with the lower and/or higher modules.

The methodology of linking any module to the system depends on how this module is used and how it relates to the other modules. For example, there is a possibility of integrating the broadcast DMIF instance with other DMIF instances to support multiple technologies within the same application. So, to give more flexibility and optimization, the
broadcast DMIF instance is linked dynamically (i.e. in a form of dynamic link library) and will be loaded only if the user needs to use the broadcast services.

Figure 4.2 Architecture of Broadcast DMIF Terminal

4.1 Server Implementation

There are number of objectives that the broadcast server implementation must fulfill. These objectives include:

- Temporal random access that enables all clients to access the media streams at any point during the service channel presentation.
- Support of multiple clients accessing multiple streams at different time frames.
- Foster the use of push channels (unidirectional channels) for data transport.
Scalability and resource balancing for managing server load and guaranteeing fair distribution of server processing amongst broadcast services.

Network constraints fulfillment through the use of Server Bandwidth Control.

The following sections describe the Broadcast server implementation and highlight the main features that leverage the system design and meet the broadcast objectives.

4.1.1 Service advertisement Layer

The service advertisement layer is responsible for sending the services descriptions to the listening clients. As shown in Figure 4.1, the layer implements an advertisement manager and announcements for all the broadcast services. The advertisement manager is responsible for creating and maintaining the announcement instances. The announcement indicates a specific manipulation to one of the broadcast channels that needs to be delivered to clients. Each announcement holds the information of specific broadcast channel, using session description protocol SDP, as well as command information for manipulating the service on the client side. Command information may include announcing new service, changing, or removing an existing one. The service manager sends the announcements to the clients through the network access medium using one of the possible transport scenarios (i.e. broadcast or multicast). The rate of sending the service announcements may be adjusted to comply with the environment conditions including network delays and bandwidth. The service manager also maintains clients' subscriptions in case of using unicast transport scenario. The service advertisement layer talks to the application layer through the Service Advertisement Application Interface (SAI). The interface provides the functionality for...
creating new announcements in case a new broadcast channel is created, changed or removed by the application as well as removing announcement in case it is expired. The interface is also used to inform the application layer of any client subscription information that should be used during service content data delivery.

4.1.2 Data Delivery layer

The main objective of the DMIF, as a service and elementary stream management layer, is to hide the details of delivery technology from the application layer. In that regard, The DMIF Application Interface (DAI) provides abstract functions to coordinate with the services and data channels currently running within the delivery layer. The DMIF Client Filter is responsible for routing the application request to the required DMIF instance that performs this request. If the DMIF instance, required to perform the application request, is not loaded, the DMIF client filter loads the required DMIF instance from the DMIF instances directory (i.e. for IM1, this directory is stored in the system registry). The interaction between the DMIF client filter and any of the DMIF instances passes through the DMIF Plug-in Interface (DPI). This interface abstracts the service functionality by allowing multiple DMIF instances from different vendors to be integrated and inter-operate to provide multiple services.

The Broadcast DMIF instance provides service, channel, and network primitives. The service primitives include attaching and detaching from a specific service. During the service attach the service acquires the initial object descriptor (IOD) which is required to start any service. The channel primitives include adding data channels based on the information in
the IOD, scene, and object descriptors and removing channels at the end of the presentation. Network primitives include adding packets to a specific network medium access to send using the required transport scenario (i.e. unicast, broadcast, or multicast).

4.1.3 Transport-level Data Channels Synchronization

One of the challenging issues that the server implementation addresses is the data channels synchronization during media transport. Although, the sync layer provides elementary streams synchronization by generating timestamp information in the slConfig (part of the SL Header, see [2] for more details) in the packet header, this information is used during channel content decoding for stream composition and playback and not during data transport. Therefore, channels content may be delivered to the clients at a higher rate than the decoding rate. Such scenario may be acceptable in some delivery technologies (e.g. in remote retrieval and local storage access). For the remote retrieval case, clients maintain a connection with the server during the entire presentation and hence may not care when the channel content has arrived. However, for stream broadcast, client may tune in to the data channel at any time during the presentation. By using asynchronous data transport, the client may lose major portions of the channel content, which should be received and decoded at the client’s listening time. If the client is tuning in at the beginning of the presentation, this might result in buffer overflow for decoding and/or composition buffers of the elementary stream on the client side in addition to possible network congestion because of sending streams at higher rates than the rate of decoding. To handle this problem, server reads the timestamps from the SL Header before the data transport and use it to delay sending the packet until the packet is eligible to be sent. The realization of this synchronization scheme is illustrated in
Figure 4.3. The server uses the “decodingTime Stamp” field from the slConfig as an indication when this packet should be decoded before being played back. Therefore, the server uses this timestamp to control the rate of sending data channel content. If we assume that the decodingTime Stamp for an AU packet with length L is T_d, sendTime Stamp is T_s. Then we can simply calculate the send time stamp as follows:

\[ T_s = \begin{cases} 
T_d - \frac{L}{C} - G & \forall T_d > (G + L/C) \\
0 & \text{Otherwise}
\end{cases} \]  

EQ 4-1

Where C is the capacity of the network link. G is a delay granularity factor that compensates for the heterogeneity of the network and congestion. While increasing G means tolerating more network delays, it might indicate that T_s will be close to zero most of the time, and this leads back to the original problem that the client may lose major portions of the channel content when tuning in at different instances during the presentation. This simple mechanism ensures the smoothness of the network traffic by controlling the stream sending rate, improves the server’s resource distribution, and enables clients to randomly access the scene content at any point during the scene’s presentation. This implementation issue will be discussed further in chapter 5.

Server also uses the accessUnitStartFlag and accessUnitEndFlag fields in the SLHeader to repacketize large AUs into small size SL-Packets to avoid client’s buffer overflow. To indicate a start of a new AU, Server sets the accessUnitStartFlag to 1 in the SLHeader and sets the accessUnitEndFlag to 1 at the end of the AU.
4.1.4 Server bandwidth control

One of the contributing issues that the framework implementation covers is the server bandwidth control. Server bandwidth control allows the server to accommodate more crucial media streams during overload conditions and compromise the network bandwidth by dropping less significant information. This increases the server scalability by supporting more crucial information in addition to preventing the network congestion.

To address this issue, an application-level, measurement-based algorithm has been devised on the server implementation to optimize the bandwidth utilization by compromising low priority streams. The algorithm assumes that the importance of the any stream within the scene context is directly proportional to the stream priority. The characteristics of this algorithm can be described as:

![Diagram of data channels synchronization]
1. **Measurement-based:** The server uses online measurements to calculate the required resources for each channel holding information of a specific elementary stream and not on a worst case scenario or resources information stored with the elementary stream object. This approach is more suitable when the channel bandwidth is fluctuating during the object composition. The server measures the instant as well as the maximum bit rate for each channel and service running on the server using the following equations.

- **Instant bit rate for channel no** $i$ or $(r_{ci})$ is defined as:

$$r_{ci} = \frac{\sum L_i}{\Delta t}, \quad \forall L_i \in \Delta t$$  \hspace{1cm} \text{EQ 4-2}

Where $L_i$ is the packet length for all the packets sent within the time period $\Delta t$.

- **Maximum bit rate for channel** $i$ (or $R_{ci}$) is defined as:

$$R_{ci} = \text{Max}\left(r_{ci}\right)_{[t_j-t_i]} \quad \forall r_{ci} \in [t_j-t_i]$$  \hspace{1cm} \text{EQ 4-3}

Where $t_j-t_i$ is the window time for calculating the max bit rate. If $t_i$ is 0, this calculates to the absolute maximum bit rate.

- **Service instant bit rate for service** $j$ (or $r_{sj}$ for $S_j$) is defined as:

$$r_{sj} = \sum_{i=1}^{n} r_{ci} \quad \forall r_{ci} \in S_j$$  \hspace{1cm} \text{EQ 4-4}

- **Service maximum bit rate** $R_{sj}$ is defined as:

$$R_{sj} = \text{Max}\left(r_{sj}\right)_{[t_j-t_i]} \quad \forall r_{ci} \in [t_j-t_i]$$  \hspace{1cm} \text{EQ 4-5}

Where $t_j-t_i$ is the window time for calculating the max bit rate. If $t_i$ is 0, this calculates the absolute maximum bit rate.

- **Total server Bandwidth** $B_t$ is defined as:
\[ B_i = \sum_{j=1}^{N} R_{ij} \]  \hspace{1cm} \text{EQ 4-6}

Where \( N \) is the no of services.

- Server residual bandwidth \( B_r \) is defined as:

\[ B_r = B_i - B_{\text{max}} \]  \hspace{1cm} \text{EQ 4-7}

Where \( B_{\text{max}} \) is the server's total bandwidth limit.

2. **Semantic-based**: The algorithm uses the stream priority as an indication of the elementary stream significance within the scene context. This means that the elementary stream with low priority will not make significant change within the scene context, and therefore the receiving terminal will not notice major change if this stream was dropped. Examples of this kind of streams may include high-resolution objects, which are encoded on multiple elementary streams with different resolution (i.e. base layer and enhancement layer(s)). The enhancement layer streams may then have low priority and server may drop them if there is no bandwidth available.

We propose two algorithms to control the server bandwidth, namely: ServicePacketDrop and ObjectPacketDrop as explained in the following section.

**4.1.4.1 ServicePacketDrop Algorithm**

Server uses this algorithm to control the total bandwidth by calculating the service packet-drop rate based on the residual bandwidth defined by EQ 4-7.
Figure 4.4 ServicePacketDrop Algorithm

Figure 4.4 depicts the flowchart for the ServicePacketDrop algorithm. The algorithm calculates the server residual bandwidth using Eq 4-7 and uses it to calculate the packet drop rate when the server is overloaded. DropRate is used by all the services running on the server to drop service AUs and lower the service bandwidth to the limit that the server can provide when it is overloaded. So, the algorithm calculates the drop rate as follows:

\[
DR = \begin{cases} 
\text{Round} \left( \frac{B_s}{B_r} \right) & \forall B_s > B_r \\
1 & \text{Otherwise} 
\end{cases} \quad \text{EQ 4-8}
\]

The condition for dropping a packet is:

\[
\text{Mod}(P_i, DR) = 0 \quad \text{EQ 4-9}
\]

Where \(P_i\) is the service or object packet index.
While this algorithm is simple and straightforward, it does not take into consideration the semantics of the services running on the server. In order to appreciate the problem of not considering the service semantics, we may summarize the potential problems with this algorithm as follows:

1. Dropping control packets: control packets are the packets that the system uses to create the required data channels to hold the incoming streams (i.e. IOD, OD, SD, packets). In broadcast scenario, these packets are repeated to accommodate “client random access” as explained before. However, dropping control packets may cause the client to incur excessive start-up delays. The high delay may cause input buffer overflow on the client side because client will not start decoding data packets until receiving the control packets required for creating the data channels.

2. Incoherent service contents: although dropping packets is the common theme for frame-based applications, it may cause incoherence in the service contents for object-based applications. Because, in object-based applications, the scene is composed in a form of objects, and each object has different characteristics based on the type of stream held within this object. Some objects like text objects (i.e. may be compressed as zip files) cannot tolerate packet loss and may waste bandwidth, if they lose packets, without contributing to the scene presentation.

Therefore, The algorithm may perform better if it becomes aware of the objects running on each service and the significance of each object within the service context.
4.1.4.2 ObjectPacketDrop Algorithm

This algorithm is used to mainly demonstrate the benefit of using object-based approach as opposed to the frame-based. The assumption in this case is that the server should serve the higher priority elementary streams under the situation of system overload. The server uses a bandwidth limit, which indicates the highest bandwidth that the server may support. This bandwidth limit may be fixed or fed to the server by external process based on the network conditions or the ISP's (internet service provider) service agreement. The algorithm sorts all channels based on the priority of the elementary streams held by each channel (see Figure 4.5). In case of system overload, the low priority channels within a specified priority level are identified, and the ones that have more bit rates are selected first for processing, so that they have more potential of reaching the bandwidth limit faster. The processing depends on the nature of the elementary stream that the channel holds. Default processing is dropping stream packets using a drop rate based on EQ 4-8 with $B_t$ equivalent to the total channels bandwidth $B_{ct}$. However, if the data channel holds a special object (i.e. Object which does not tolerate data loss), server drops the whole stream by suspending the data channel transfer. The detailed flow of actions is depicted in Table 4.1. Figure 4.6 represents the algorithm's flow chart showing all the logical paths and the action performed for each path.

![Figure 4.5 Data channels sorting](image)

63
sorted data channels

A

calculate residual bandwidth

Server overloaded? residual band>0

Yes

For all the channels in the priority level P

Select channel with maximum bit rate

Is special object channel?

Yes

Suspend channel

Update residual bandwidth

No

add to total channels bandwidth

Calculate DropRate

Increment P

P> priority threshold?

Yes

DropRate=1

Update system status to overloaded

A

residual bandwidth < total channels bandwidth

No

Update system status as normal

For all the channels in the priority level P

DropRate>0?

Yes

Resume a channel

Send Droprate to channels

Decrement P

A

Figure 4.6 ObjectPacketDrop Algorithm
<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Check the server bandwidth and calculate the residual bandwidth</td>
</tr>
<tr>
<td>2</td>
<td>If residual bandwidth is greater than zero (overload), go to step 9</td>
</tr>
<tr>
<td>3</td>
<td>Set the system status to &quot;Normal&quot;</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>If the drop rate is greater than zero (there are still channels with packet dropping), go to step 7</td>
</tr>
<tr>
<td>6</td>
<td>Resume channels within this priority level from the list of suspended channels, then go to step 8</td>
</tr>
<tr>
<td>7</td>
<td>Set drop rate to zero and send command to the selected channels within this priority level P to set drop rate to zero</td>
</tr>
<tr>
<td>8</td>
<td>Decrement P, then go to step 1</td>
</tr>
<tr>
<td>9</td>
<td>For all the channels in the priority level P, do the following</td>
</tr>
<tr>
<td>10</td>
<td>Select channel with maximum bit rate from the list of unselected channels</td>
</tr>
<tr>
<td>11</td>
<td>If the channel holds a special object, then suspend the channel</td>
</tr>
<tr>
<td>12</td>
<td>Update residual bandwidth, and go to step 1</td>
</tr>
<tr>
<td>13</td>
<td>Add the channel bandwidth to total channels bandwidth ($B_{ct}$)</td>
</tr>
<tr>
<td>14</td>
<td>If the residual bandwidth is less than the total channels bandwidth ($B_{ct}$), go to step 19</td>
</tr>
<tr>
<td>15</td>
<td>Increment the priority P</td>
</tr>
<tr>
<td>16</td>
<td>If P is greater than the priority threshold ($P_1$), continue, otherwise go to step 9</td>
</tr>
<tr>
<td>17</td>
<td>Set the drop rate to 1 (suspend the selected channels)</td>
</tr>
<tr>
<td>18</td>
<td>Update system status to &quot;overloaded&quot; so that no further services will be admitted, then go to step 1</td>
</tr>
<tr>
<td>19</td>
<td>Calculate drop rate using EQ 4-8, replacing $B_t$ with $B_{ct}$</td>
</tr>
<tr>
<td>20</td>
<td>Send drop rate to all the selected channels within the priority level P, then go to step 1</td>
</tr>
</tbody>
</table>

**Table 4.1** ObjectPacketDrop flow
4.1.5 Multiple service application sub-layer

Another issue that the server implementation addresses is handling multiple services on the application sub-layer. This issue was raised as a result of a specific limitation with IM1 handling multiple services. Referring to Appendix A, the application layer interacts with IM1’s system layer through the executive module, which acts like the system brain. The executive component controls the execution of the system, instantiates other modules, and handles the data flow between the modules. This executive module has appeared to be unable to load and handle more than one service. Therefore, a new application layer interface has been created through which the application talks to IM1 system layer. Figure 4.7 shows where the multiple service layer is located within the system context.

![Diagram showing the multiple service layer](image)

**Figure 4.7  Multiple service layer**

This layer is developed to hold the data structures needed to handle more than one service within the broadcast system. The multiple-service executive is responsible for maintaining the service components during the presentation. Figure 4.8 shows the detailed components of the multiple-service layer and how the services components are connected with the multiple-service executive. The multiple-service executive module also defines an interface for the application to create new services and redirect the application calls to IM1 system core while maintaining the service components for each service. Due to the level of
control that the multiple-service executive has on the system, the server bandwidth control
group is built within the Multiple-Service Executive to be able to control the current
running services and channels. For more information about IM1 system Architecture, see
Appendix A.

![Diagram](image)

**Figure 4.8** Multiple service layer modules

### 4.2 Client Implementation

In broadcast scenario, clients do not have to fetch the information servers. As a
result, thin clients are normally a better and reasonable choice for broadcast applications.
Such clients act as dummy listeners to the stream contents that server sends. So, the logic
required to handle service advertisement and data delivery should be marginal compared to
the logic on the server.

#### 4.2.1 Service Advertisement module

Client implementation is based on the same architecture depicted in Figure 4.1. The
advertisement manager module (AM) is responsible for creating and maintaining the required
SAP announcements on the client side. AM dispatches a network listener at start-up to listen to the announcements sent by the server. Based on the received announcements, the AM manipulates (i.e. adds, removes, or modifies) the relevant SAP announcement object. If the user requires subscribing to one of the available services, a timed-out request is sent to the server with information about the required service. If a positive response was received from the server, the AM invokes the DMIF instance(s) required to handle the service. This happens through the SAI interface, which provides the requests to the application layer to instantiate one DMIF instance per each transport scenario used within the service (i.e. unicast, broadcast, or multicast).

4.2.2 DMIF delivery layer module

The DMIF delivery module implements the recommended DMIF Application interface (DAI), which hides the delivery technology from the application. The DMIF client filter maintains the routing of the application requests to the target DMIF instances through the DPI interface. The DMIF broadcast service is responsible for invoking the service, channel, and network primitives as explained in section 4.1.2. The data channel module filters the incoming stream to extract the packets that belong to this data channel and forward them to the higher layers for processing. The data channel module also filters the repeated packets by using the packet ID field in the packet header as described in section 3.2.5. The network access module is responsible for streaming the incoming packets and verifying the packet header for error recovery.
Chapter 5  Performance analysis

This chapter provides the performance analysis for the broadcast system based on our client/server implementation discussed in Chapter 4. This performance analysis lays the basic steps of verifying the system functionality and the correctness of the system implementation. The broadcast system implementation has been tested using the following configuration:

- The server and all clients are running on the same local area network.
- Server is running on a single CPU PIII 730Mbps with 128MB RAM using Windows NT workstation.
- Clients are running on the same type of machines using Windows NT, windows 98, or windows 2000 OSs.
- For each experiment a reporting code was implemented to calculate, and report the information needed (e.g. channels instant bit rate, decoding time stamps, etc.).

Table 5.1 lists the main parameters for the broadcast system implementation and the value used for each parameter.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Value</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Delay granularity</td>
<td>100 msec</td>
<td>Section 4.1.3</td>
</tr>
<tr>
<td>$\Delta t_i$</td>
<td>Instant bit-rate delta</td>
<td>200 msec</td>
<td>EQ 4-2</td>
</tr>
<tr>
<td>$</td>
<td>t_j - t_i</td>
<td>_{\text{max}}$</td>
<td>Maximum bit-rate interval</td>
</tr>
<tr>
<td>$P_d$</td>
<td>Default UDP port number</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>Bandwidth control loop delay</td>
<td>500 msecs</td>
<td></td>
</tr>
<tr>
<td>$P_t$</td>
<td>Priority threshold for ObjectPacketDrop algorithm</td>
<td>2</td>
<td>Figure 4.6</td>
</tr>
</tbody>
</table>

**Table 5.1 List of parameters**

### 5.1 Transport-level data channels synchronization

The purpose of this experiment is to measure the effect of the elementary stream transport synchronization on the stream content data delivery. Section 4.1.3 discussed the issue using data channels synchronization to accommodate the client random access and to smooth the network traffic. In this experiment, the "Linda" stream was used with an average bit rate of 140kb/s, frame rate of 15-frames/sec and stream duration of almost 5 minutes. Figure 5.1 shows the instant bit rate of the "Linda" stream with and without data channel synchronization. For the unsynchronized case, the entire stream was transferred within the first 2 seconds of the scene presentation with a maximum bit rate of 4.6 Mbps. In some cases, assuming that the client has the proper buffers for network access and stream decoding, the client may be able to receive and playback the scene content if the client was tuning in at time zero. The reason for that is that the synch layer will synchronize the scene elementary streams for decoding and playback. Although, in terms of performance, this scenario is obviously not acceptable, it may be acceptable in some applications that maintain connections between
server and clients throughout the entire presentation. However, in broadcast applications, this scenario is not acceptable because of the client random access support. So, for case a) if the client started listening to the broadcast channel at time 3, it will lose all the stream content. Also, in case of using multiple audio-visual elementary streams, the client may lose different portions of each elementary stream, which makes the streams out of synchronization on the client side.

Figure 5.1 Instant bit rate for "Linda" video stream
If we denote the decoding timestamp by \( T_d \) and receiving time stamp on the client by \( T_r \), we need to make sure that \( T_r \) will always be less than \( T_d \) to minimize the packet jitter on the client side. Figure 5.2 shows the decoding time stamp and the receiving time stamp on the client side as a result of receiving the “Linda” stream. The decoding time stamp is calculated on the client side based on main client object’s clock time. The object’s clock may be used for multiple elementary streams to indicate that the elementary streams are coming from the same encoder source and they should be synchronized. The clock in this case is reset by the arrival of the first SL-Packet of the elementary streams by assuming that this packet is always received on time or \( T_d = T_r \) on service start-up. This concept is called intra-stream synchronization or synchronizing streams coming from the same encoder source and the service start-up time will be called object time base or OTB. The more complicated case is to receive and compose multiple elementary streams coming from multiple encoder sources. In this case, the client may use multiple clocks to represent the OTB of each clock. The OTBs are calculated based on object clock reference provided by the encoder source.

As part of the service advertisement layer, the client recognizes the required network listeners needed to receive the incoming stream. The client then dispatches the network listeners and allocates the required network access buffers to store the incoming packets. The client begins the presentation by waiting to read, and decode, the IOD, SD, ODs, before creating the data channels required to receive the stream content (SL packets). Although, the client may receive the SL packets before the object descriptors, it starts decoding the SL packets after decoding the object descriptors. This while imposes more delays on the client at start-up, it helps minimizing the packet jitter since the client will have multiple SL packets.
waiting in the network buffer, ready for composition. This is illustrated in Figure 5.2 by the flat line at the beginning of the receiving time stamp $T_r$.

**Figure 5.2 Decoding time stamps and Receiving time stamps**

From the discussion in section 4.1.3 we stated on the server side that:

$$T_s = \begin{cases} T_d - \frac{L}{C} - G & \forall \ T_d > (G + \frac{L}{C}) \\ 0 & OW \end{cases}$$

EQ 5-1

Now, we can rationally assume that the granularity factor $G$ has no effect on the shape of the curve in Figure 5.2. The reason for this is that $G$ imposes temporal shift for the entire stream content and not for portions of the stream. Hence, $G$ may affect the calculation of OTB on the client side, but it has no effect on the packet jitter. Using high positive values for $G$ may cause the client to lose portions of stream content while using high negative values for $G$ may make the client compose the stream content within different time frame than the time the stream was being broadcasted. Therefore, $G$ may have a recommended range of
values of [2000, -2000] milliseconds on LAN environments. However, it's recommended to use positive value for G in other network environments (e.g. wireless networks).

The channels synchronization mechanism also improves the fairness of scheduling the server resources amongst the active services. Since the server schedules the time to send the service AUs relative to its decoding time stamp, this allows the server to partition the available resources, including CPU time and network bandwidth, amongst the active services running on the server at any point in time. To illustrate this point we loaded the server incrementally by 16 services and tested the effect of channels synchronization on the resources consumed by each service. For this purpose, we define the instant average bit rate for a service $S_i$ at any time $t$ as follows:

$$R_i(t) = \frac{\sum_{t=0}^{t} L(t)}{t}$$  

Where $L(t)$ is the SL packet length for all the service AUs sent until time $t$. This bit rate represents the rate of the service information delivered at any time. Using this equation, the bit rate for the Linda stream is shown in Figure 5.3. For the unsynchronized case, the instant average bit rate grows rapidly at the beginning taking all the resources required to deliver the entire stream in a short period of time. Then, it moves down as the stream expires. In the synchronized case, however, the average bit rate grows gradually taking only resources as needed. As a result, server can allocate resources to more services.
Figure 5.3  average bit rate for “Linda” audio-visual stream

Figure 5.4 shows the results of running 16 services on the server with and without synchronization. Services struggle for the server resources in case a). When server resources are used up, the remaining services wait for the resources to become available. Figure a) shows the contention period when all the server resources are used up. In case of b), server
allocates resources to services as needed. We can see, in this case, the amount of information delivered for all services is fairly distributed.

Figure 5.4 Instant average bit rate for 16 services
5.2 Channels overhead Vs client start-up delay

The purpose of this experiment is to measure the channels overhead and determine the effect of repeating the content of the object descriptor channels on the client start-up delay during the service start. The average overhead caused by any channel at any time during the presentation can be simply defined as:

$$V_c(t) = \frac{\sum_{\tau=0}^{t} L_c(\tau)}{\sum_{\tau=0}^{t} r_c(\tau)}$$  \hspace{1cm} \text{EQ 5-3}

Assuming $\Delta t$ is constant for calculating instant bit rates, $L_c$ and $L_s$ are packet lengths for packets transferred until time $t$ for channel $c$ and service $s$ respectively, $r_c$ and $r_s$ are the instant bit rate for channel $c$ and service $s$ respectively as defined by EQ 4-2. Results show that the overhead starts with high values at the beginning of the presentation as channel $c$ content is transferred and decoded at the beginning before sending audio-visual streams contents. When the transport of the audio-visual content starts, the overhead tends to stabilize around the typical value of the average overhead caused by channel $c$.

![Figure 5.5 Average overhead over time](image-url)

Figure 5.5  Average overhead over time
In section 3.2.3, we calculated analytically the average overhead caused by sending the IOD, SD, and ODs using equations EQ 3-4 through EQ 3-7. To generalize these equations, we can assume that the average overhead caused by data channel C running within multiple services is calculated as follow:

\[ V_c = \frac{N L_c}{T_c \sum_{i=1}^{N} R_i} \]

EQ 5-4

Where \( L_c \) is the length of all packets sent on channel c during period of time \( T_c \), and \( R_{si} \) is the average bit rate of service no i, and \( N \) is the number of services running on the server. Figure 5.7 shows the average overhead versus the no of services as a result of loading the server with 20 services incrementally and measuring the overhead after creating each service. The average bit rate for each service is around 140Kbps. It's clear from the figure that overhead is almost constant as the no of services running on the server increases, which conforms to the analytical results driven in section 3.2.3. Even though the experiment was performed on relatively low bit rates, the OD channel overheads appeared to be insignificant and can be tolerable on any network platform. This is, of course, more compelling for high bit rate services, as the overhead should be negligible.

The client start-up delay may be calculated analytically based on the channel's repetition period and the decoding time. Assuming that the period of repeating the content of channel i is \( P_{ci} \) (i.e. assuming that the network delay will almost be constant for all channels contents. So, the effect of the network delay is negligible) and time required for decoding the channel content on the client side is \( D_{ci} \). So, the total time required for the client to receive
and decode the channel content is \( T_{ci} = P_{ci} + D_{ci} \). Because the client is not receiving the repeated channels sequentially, there may be an overlap between the time-periods required to receive the repeated channels. If we have \( M \) repeated channels, the client will be ready to receive the scene content after receiving the contents of all the repeated object descriptor channels. So, the total delay required to receive all the repeated channels’ contents \( (T_i) \) will be a uniformly distributed random variable with a minimum value of \( \max(T_{ci})^M \) and a maximum value of \( \sum_{i=1}^{M} T_{ci} \). If we assume the simple case of equal \( T_{ci} \)’s, then the minimum value of \( T_i \) would be \( T_c \) and the maximum value would be \( MT_c \). For example, for the overhead problem in Figure 5.5 and Figure 5.7, we have to repeat 3 channels (i.e. IOD, SD, and OD channels), with a period of 500 ms for repeating the individual channel content. So, the maximum \( T_i \) is 1.5 seconds neglecting the time required to decode the individual channel content. During this time \( T_i \), client may receive SL packets for audio-visual elementary streams, but no SL packets will be decoded until the client receives the OD channel contents and this, while imposes delays, it allows the client to tolerate packet jitter since the SL packets wait for decoding during this time as explained in section 5.1.

The client start-up delay also affects the client buffer size. The client terminal has a finite buffer to store the received data until they are decoded and played back. All the SL packets received during the client start-up time will be stored in the client’s buffer waiting to be decoded and played back. The amount of data received during client start-up depends on the type and number of elementary streams being sent. Since there are no limits on the number of objects in audio-visual presentations, it is not practical to have sufficient buffer for all presentations. So, the buffer size on the client’s terminal should be designed to support a
class of presentations. The following equation provides an estimate for the client network buffer size $b_{min}$:

\[ b_{min} = T_s B_s \]

EQ 5-5

Where $B_s$ is the maximum service bit rate on the network link and $T_s$ is the total startup delay.

---

**Figure 5.6** Client startup random variable

**Figure 5.7** Average overhead versus number of services
5.3 Server Bandwidth Control

The server bandwidth control algorithm is explained in section 4.1.4. The algorithm is used to restrict data channels' and services' bit-rates to fulfill the server's bandwidth limit. Figure 5.8 and Figure 5.9 show the channel and service instant and maximum bit rates for "Linda" audio-visual stream based on EQ 4-2 through EQ 4-7. 100ms, and 2seconds, time windows were used to calculate the instant and maximum service's and data channel's bit-rates respectively.

The following sections show the results of applying the ServicePacketDrop and ObjectPacketDrop algorithms to control the server bandwidth.

![Graph showing data channels' instant bit-rate for "Linda" audio-visual stream]

Figure 5.8 Data Channels' Instant bit-rate for "Linda" audio-visual stream
Figure 5.9  Service instant/maximum bit-rate for “Linda” audio-visual stream with 2 seconds window size

5.3.1 ServicePacketDrop Algorithm

This algorithm does not take into consideration the semantics of the service scene and it is not priority based. Although, the algorithm guarantees the maximum server bandwidth as shown in Figure 5.10 through Figure 5.15, the non-priority feature may cause large startup delays when the client is unable to receive the object descriptors during the service start-up.
Figure 5.10  “Linda” audio-visual stream with ServicePacketDrop (1)

Figure 5.11  “Linda” stream a) No Bandwidth Control b) with 100Kbps ServicePacketDrop

Figure 5.11 shows one of the presentation frames with no bandwidth control and with 100Kbps ServicePacketDrop bandwidth control algorithm. The drop rate reported at the time instant of this frame is 3. This means that among every 3 packets the server has to drop one packet to fulfill the network bandwidth limit.
Figure 5.12  "Linda" audio-visual stream with ServicePacketDrop (2)

Figure 5.13  8-Object stream with ServicePacketDrop Algorithm (1)
Figure 5.14  8-Object visual stream with no bandwidth control

Figure 5.15  8-Object stream with ServicePacketDrop Algorithm (2)
Figure 5.16 8-Object stream with 800Kbps ServicePacketDrop

Figure 5.16 shows one frame of the 8-object stream presentation at time almost 19 seconds. The figure shows the packet loss happening in all the objects at the same rate to fulfill the network bandwidth limit. The drop rate at the instant of time of this frame being transferred is 6. This means that the server drops one packet every 6 packets being transferred from the presentation regardless of any priority scheme for the audio-visual objects.

From Figure 5.10 through Figure 5.16, the algorithm limits the duration of the service burst (i.e. service rate beyond the maximum bandwidth) to be close to the duration of calculating the service maximum bit rate (i.e. \([t_f-t_i]\) in EQ 4-3), which is 2 seconds in this experiment. However, it may be noticed that the burst size may sometimes exceed the 2 seconds period. This may be caused by the fact that the server’s loop frequency used to calculate the server and the residual bandwidths is not in sync. with the 2 seconds maximum service rate period (see Figure 5.17). Another cause may be that the service rate takes time to
follow the maximum bandwidth limit. So, to help the server to reach the required maximum bandwidth in a short period of time, the loop frequency that the server uses to calculate the server and residual bandwidths should be small compared to the service maximum bit rate time period.

As explained in section 4.1.4.1, the algorithm may cause excessive client start-up delays, client buffer overflow, and incoherent service contents. The first two problems may happen as a result of dropping the control packets that the client uses to create the data channels required to handle the incoming service streams. Incoherence in service contents is caused by dropping packets from special objects, which is explained, in section 4.1.4.2.

![Figure 5.17 Timing for server’s bit rates calculations](image)
5.3.2 ObjectPacketDrop Algorithm

Server uses this algorithm to suspend or drop packets from low priority objects during system overload. The algorithm uses the scene semantics to control the bandwidth. The algorithm sorts the opened channels based on the priority of the elementary stream content held by the channel. On server overload, the system compromises the bandwidth by suspending the low priority special objects and/or by dropping packets held by low priority objects. Figure 5.18 and Figure 5.19 show the results of applying the ObjectPacketDrop algorithm on the same 8-object stream. In this case, the object descriptor streams may take high priority to assure the minimum client start-up delay.

Figure 5.18 8-object stream using ObjectPacketDrop algorithm (1)
Figure 5.19 8-object stream using ObjectPacketDrop algorithm (2)

Figure 5.20 8-Object stream with 800Kbps ObjectPacketDrop

Figure 5.20 shows one frame at the same time instance as Figure 5.16 after applying ObjectPacketDrop algorithm. The figure shows the packet drop happens only in the object
streams marked by white circles. To fulfill the network bandwidth limit, the drop rate within the 4 objects is 2 at almost 19 seconds time instance, which means that the server drops one packet out of every two packets to fulfill the bandwidth limit.

The ObjectPacketDrop algorithm guarantees the bandwidth limit while taking into consideration the significance and the special characteristics of each object within the scene context. This helps correcting the problems caused by the ServicePacketDrop algorithm including high client start-up delays, client buffer overflow, and incoherent service contents. Also, in case of using special objects, the server stops/suspends the special objects while the algorithm tries to reach the maximum bandwidth limit, which means that the service bit rate may drop faster than the previous case. Thus, the algorithm may perform faster in reaching the required maximum bandwidth.

Although, both algorithms, in general, prevent the high service bursts and fulfill the network bandwidth constraints, they do not shape the service traffic to utilize the low bit rate periods. This may be enhanced using either a traffic shaper (e.g. token bucket) or object-based scheduler (e.g. Fullsched, GapSched [15]). The later is more recommended since it schedules the service packets based on deadlines (decoding timestamps). However, the scheduling algorithm may be customized to benefit from the presentation semantics (e.g. Object scheduling like IncSched [15]).
Chapter 6  Conclusions and Future Work

This thesis reported the architecture, design, and implementation of a novel framework for MPEG-4 media streams broadcast. The framework promotes the object-based concept for streams composition, and the push model for broadcast information distribution. The framework proposed solutions to some issues associated with realizing the MPEG-4 media streams within the broadcast applications. The framework defined the methodology and techniques for managing multi-service broadcast streams over different network platforms.

The real implementation plus the performance results verify the visibility of the framework over IP networks, and uncover the strengths of using the object-based approach for streams composition and transport. This object-based concept while adds complexities to the system regarding the scene description and elementary streams synchronization, it helps optimizing the quality of service based on the characteristics of each object within the scene presentation.

The adoption of the push model for broadcast service advertisement and data delivery helps minimizing the network flow for data signalling and control. Hence, clients only listen to the network streams at their convenience and do not require to fetch or request connection with the server during the service presentation. This makes the client logic very simple and visible to be implemented, in conjunction with the stream decoders, on special devices (e.g. handheld devices, PDAs, cellular phones). Besides, due to the minimum amount of information in the upstream direction (i.e. from client to server), the framework may also be used in network platforms where network delay is somewhat expensive like satellite and
wireless networks or special types of asynchronous network connections like ADSL (Asynchronous direct subscriber line).

DMIF standard provides a sub-framework for elementary streams management in addition to hiding the delivery technology details from the DMIF user and ensuring the interoperability between end-systems. The specifications are still immature for broadcast scenario, particularly regarding the definition of signalling and network interface to facilitate broadcast-specific features such as Client Random Access and elementary streams’ resource distribution. This immaturity added some challenge within the system design for devising some mechanisms for packetization and overhead control that accommodates such features while still optimizes the network flow and preserves the server scalability.

Server bandwidth control is another mechanism that serves the scalability of the streaming server by increasing the number of crucial data channels that the server can accommodate during the network overload. The algorithm preserves the scene semantics by prioritizing audio-visual objects according to their significance within the scene context. The algorithm also controls the traffic bursts while keeps the scene contents homogeneous, by classifying objects based on their characteristics and compromising server bandwidth accordingly by suspending objects or dropping objects’ packets.

Using IM1 reference software challenged the system design and implementation to overcome some limitations regarding application layer mutli-service handling and data flow control within the DMIF instance. The first was overcome by implementing a multi-service layer on top of IM1 application layer to allow the co-existence and control of multiple services within the application layer and to derive the IM1 system layer to create and manage
the services and their respective DMIF instances. The data flow control uses the composition
timestamp to schedule the SL packets sending to control the flow rate from the server. It also
prevents network congestion and enables clients to randomly access the scene content at any
point during the scene presentation.

6.1 Future work

The framework constructs some main pillars that may sustain future work in the area
of mpeg-4 streams broadcast over different network platforms. Further enhancements may be
investigated and applied in several areas within the system. Server enhancements may
include the development of object-based scheduling algorithm that optimizes the bandwidth
utilization by scheduling the SL packets within DMIF layer based on their deadlines (i.e.
composition time stamps) and according to the network QoS parameters for each elementary
stream including delay, bandwidth and jitter. Another enhancement to the server may be done
by investigating the possibility of developing data fragmentation techniques, and broadcast
strategy, for video-on-demand applications [17,18]. Future studies can also investigate the
service fast start mode or sending all the required object descriptors to clients in one SL
packet to minimize the client start-up delay and client buffer size. Such enhancement is not
possible conceptually using the current DMIF standards because the standards separate the
initial object descriptor from elementary streams object descriptors. However, This
mechanism maybe crucial for some network-platforms where delay is a critical parameter
(i.e. satellite networks).

Current server implementation does not support different client capabilities. For
example, clients use different bandwidth capabilities for receiving the service content. Such
problem maybe investigated to try to architect an intermediate gateway, which is used to scale the service contents before reaching the clients. This can help preventing clients with low network connections from incurring high delays or not being able to cope with the flow rate being sent from the server. The gateway can use again some of the techniques explained in this thesis (e.g. ServicePacketDrop, ObjectPacketDrop) to scale the service contents based on each client's capabilities.

Client implementation may also be enhanced by implementing caching for receiving service advertisements from the server. This caching scheme reduces the client start-up delay before the client can receive all the service announcements that the server sends. Instead, the client caching is used to store the incoming announcements and client can access the announcements from the cache upon startup.
Bibliography


Appendix A. IM1 System Background

As we will discuss in later chapters, our system implementation is based on the IM1 implementation software implemented by the MPEG committee towards MPEG4 standards verification. In this chapter, we give an overview about IM1 reference implementation as an introduction to our system architectural design.

IM1 is one of two main groups (i.e. implementation software 1 and 2 or IM1 and IM2 for short) formed by the MPEG committee in 1997 to provide a complete solution for MPEG4 players, which should be able to prove that the MPEG4 standards work. IM1 group proposed a C++-based implementation, whereas IM2 would use Java. From those two groups, only IM1 produced several MPEG-4 players. One of them is a 2 dimensional player, which handles the 2D profile of the standard. Figure A.1 depicts the main IM1 system modules and the location of each module within the system architecture. The following section details the functions of each module within the system context.

A.1 IM1 system architecture

The IM1 platform comprises several modules each developed and tested independently. For simplifying the perception of these modules, we classify them based on the layer that each module belongs to. The main modules in the application layer include:
A.1.1 The frame application - Application layer

This is the interface through which the user interacts with the players. It provides the frame for the scene view and handles user interaction operations such as mouse commands and user commands (i.e. play, pause, stop, etc.). The interface may also include some DMIF related functions for each delivery technology. For example, open service using URL, enables the user to access specific service from the server using the specific URL format associated with each delivery technology. The “display available channels” function in case of broadcast scenario enables the client user to display all the available services that are being broadcast by the server at any time.
A.1.2 The Core - Application layer

This module includes sub-modules for controlling the execution of the player (the Executive), sync layer (SL) processing, buffer and time management, Binary Format for Scenes (BIFS) and object descriptor (OD) decoding, and the creation and manipulation of the scene tree. The core components include:

- **The Executive**: This component controls the execution of the player, instantiates other modules, and handles the data flow between the modules.

- **The Sync layer manager**: This component receives (through DMIF) data packets of elementary streams, decodes SL information, and dispatches access units to decoding buffers.

- **The buffer manager**: Controls the flow of data through buffers. The buffers used by the system are time stamped so that units may be fetched from buffers at specific time. Hence, this module manages both memory and synchronization.

- **The OD decoder**: Parses the OD streams and passes the extracted data to the Executive for execution.

- **The BIFS decoder**: parses scene description streams, creates the scene tree, and updates it according to BIFS-update and BDS-animation commands included in these streams.
- The scene Manager: This module provides the infrastructure for the scene graph operation. It handles most of its generic actions, such as processing of routes, firing of events when event-out fields are modified, notification to the node when any of its fields are changed, and execution of conditional and inline nodes. It also implements basic functionalities of a few classes that are used as base classes for specific nodes.

A.1.3 The compositor - Application layer

This module is the “consumer” of the scene tree. It continuously traverses the tree, fetches composition units from composition buffers, and composes and renders the scene.

A.1.4 The decoders - Application layer

These are group of modules that read access units from decoding buffers, decompress them. And store the decoded data in the composition buffer.

A.1.5 IPMP systems - Application layer

Optional modules that provide intellectual property management and services protection. IPMP systems may require pre-decoding (e.g. decryption) or post-decoding (e.g. watermarking) processing.
A.1.6 DMIF Stack – session and transport layers

The DMIF instance packaged with IM1 reference software supports only the local storage scenario where the media streams are stored and displayed on the local machine and no network access is required. The local storage instance includes the standard DMIF application interface DAI, Flexmux, and transmux sub-layers as follows:

- **The DMIF-Application Interface:** The DMIF-Application Interface (DAI) is a semantic API that allows the development of applications transparent to the supported delivery technologies. By using the DAI, an application could seamlessly access content from broadcast networks, from local storage devices and from remote end-systems without being aware of the network and transport details. Through the DMIF-Application Interface DMIF Users are hidden from the delivery technology details (for both Data and Control Planes), and just manipulate Service Sessions and channels. For this purpose, the DMIF-Application Interface defines service, channel, and data primitives for controlling information flow from the application to the DMIF and transport layers.

- **FlexMux layer:** The use of the FlexMux layer is optional. If it is bypassed, data streams may be directly carried over a TransMux channel without additional overhead. The FlexMux represents a multiplexing layer that provides interleaving of one or more data streams belonging to different sources with similar QoS requirements. Therefore, applications may bypass or redefine the FlexMux scheme based on the system design for network access and data transport. For example, our broadcast system uses different
multiplexing scheme using elementary stream identifiers as will be described in the next chapter.

- **Transport or TransMux layer:** The TransMux layer provides a reliable error detection and framing of data streams. The generic term TransMux layer is used to abstract any transport protocol stack that is suitable to transport data streams and fulfill the end-to-end QoS requirements. The selected transport stack should provide appropriate framing of FlexMux streams, error detection and, if possible, delivery of corrupted data along with an error indication.
Appendix B. Broadcast/Multicast Background

B.1 Sending/Receiving an IP broadcast Datagram

To send a broadcast datagram on Windows NT and Windows 2000, specify an IP broadcast address 255.255.255.255 in a sendto function call and any port number you wish to broadcast data into. During socket initialization, use the setsockopt function to set options for broadcast as follows:

```c
BOOL fBroadcast = true;
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (CHAR *) &fBroadcast, sizeof(BOOL));
```

To receive IP multicast datagram, receiver uses the same port used during data sending in a bind function:

```c
bind(sock, (SOCKADDR *) &sockAddr, sizeof(sockAddr));
```

B.2 Sending/receiving an IP Multicast Datagram

To send a multicast datagram on Windows NT and Windows 2000, specify an IP multicast address with a range from 224.0.0.0 through 239.255.255.255 as the destination address in a sendto function call. During the socket initialization, use the setsockopt function to set options for IP multicasting. By default, IP multicast datagrams are sent with a Time to Live (TTL) of 1, which prevents them from being forwarded beyond a single sub-network. The following code example shows how to change this.

```c
int ttl = 1; // Limits to local subnet.
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL,
```
Multicast datagrams with a TTL of 0 are not transmitted on any sub-network. Multicast datagrams with a TTL of greater than 1 may be delivered to more than one sub-network, if there are one or more multicast routers attached to the first sub-network. A multicast router does not forward multicast datagrams with destination addresses from 224.0.0.0 through 224.0.0.255, inclusive, regardless of their TTL value. This address range is reserved for routing protocols and other low-level, topology discovery protocols, or maintenance protocols. These include gateway discovery protocols and group membership reporting protocols.

For receiving multicast datagram, receiver uses the same port in a bind function:

```c
bind(m_socket,(SOCKADDR *)&m_socketAdd, sizeof(SOCKADDR_IN));
```

Receiver, then, subscribes to the required IGMP group address using the `setsockopt` function:

```c
setsockopt(m_socket,IPPROTO_IP,IP_ADD_MEMBERSHIP,
(char *)&mreq,sizeof(mreq))
```
Appendix C. DMIF URL Format

C.1 Generic concepts

A URL contains the name of the scheme being used (\textit{<scheme>}) followed by a colon and then a string (the \textit{<scheme-specific-part>}) whose interpretation depends on the scheme.

\textit{<scheme>:<scheme-specific-part>}

The local DMIF parses the \textit{<scheme>} part of the URL in order to identify and activate the appropriate DMIF Instance. The local DMIF Instance in turn parses a portion of the \textit{<scheme-specific-part>} in order to establish a session with the designated remote (or emulated remote in the case of local storage and broadcast scenarios) DMIF; the remote (or emulated remote) DMIF parses a further portion of the \textit{<scheme-specific-part>} to identify the remote application program; the remote (or emulated remote) application program parses the remaining portion of the \textit{<scheme-specific-part>} to identify and activate the desired service.

C.2 URL schemes allowed in DMIF

The following are the URL schemes currently allowed in this part of ISO/IEC 14496:

"dudp:"
"dtcp:"
"datm:"
C.3 New URL schemes

Note:

The URL schemes specified here are new schemes, which are not yet officially registered at the time when this Appendix is written. URLs for experimental schemes may be used by mutual agreement between parties. Scheme names starting with the characters "x-" are reserved for experimental purposes. In time the URL schemes here specified shall be registered with the Internet Corporation for Assigned Names and Numbers (ICANN) that maintains the registry of URL schemes. As long as the URL schemes here specified are not officially registered, characters "x-" have to be added in front of it.

C.3.1 URL scheme for DMIF signalling over UDP/IP

The URL for default DMIF signalling over UDP/IP follows the generic syntax for new URL schemes defined in RFC 1738. Usually the URLs that involve the direct use of an IP-based protocol to a specified target on the Internet use a common Internet specific syntax:

```
//<user>:<password>@<host>:<port>/<url-path>
```

The URL for default DMIF signalling over UDP/IP consists of the following:

```
x-dudp://<user>:<password>@<target dmif>:<dmif port>/<url-path>
```

The `<host>` is the target DMIF and hence for the "x-dudp:" scheme it will be named `<target dmif>`. 
The \textless port\textgreater indicates the UDP/IP socket port number over which the DMIF signalling messages will be delivered and hence for the "x-dudp:" scheme it will be named \textless dmif port\textgreater.

\textbf{C.3.2 URL scheme for DMIF signalling over TCP/IP}

The URL for default DMIF signalling over TCP/IP follows the generic syntax for new URL schemes defined in RFC1738. Usually the URLs that involve the direct use of an IP-based protocol to a specified target on the Internet use a common Internet specific syntax:

\texttt{//<user>:<password>@<host>:<port>/<url-path>}

The URL for default DMIF signalling over TCP/IP consists of the following:

\texttt{x-dtcp://<user>:<password>@<target dmif>:<dmif port>/<url-path>}

The \texttt{<host>} is the target DMIF and hence for the "x-dtcp:" scheme it will be named \texttt{<target dmif>}. The \texttt{<port>} indicates the TCP/IP socket port number over which the DMIF signalling messages will be delivered and hence for the "x-dtcp:" scheme it will be named \texttt{<dmif port>}. 
Appendix D. Client and server user interface

The objective of this Appendix is to describe the basic user interface functionality for the broadcast system implementation.

D.1 Server user interface

Server user interface has two main functionalities for creating new services based on local MPEG4 streams and monitoring running services by displaying a down-sampled version of the running streams for monitoring.

D.1.1 Create new Service

Server admin uses this function to create a new service based on a stored MPEG-4 file. After the local MPEG-4 file is opened, server shows the service information panel depicted in Figure D.2. User may modify any of SDP information displayed based on the environment, then click OK to create the new service.

![Figure D.2 Service description based on SDP information](image-url)
E.1.2 Monitor running services

User may, at any time during the service presentation, use this function to display a down-sampled version of all the services currently running on the server. User may interact with the running services by double clicking on one service, and a full version of the required service will be displayed.

![Server user interface](image)

**Figure D.3** Server user interface

D.2 Client user interface

Client user interface has the several functionalities for manipulating the running service presentations including change frame rate, pause/resume, open new local MPEG-4 file, open URL. After configuring the client access a broadcast streaming server, client user may display all the services (broadcast channels) available on the server and browse the
information for each service as shown in Figure D.4. Client user may then start listening to one broadcast channel by clicking on subscribe or stop listening by clicking on unsubscribe.

Figure D.4  Client user interface