MASTER

Design and realization of a video-on-demand demonstration system

Mulder, A.

Award date:
1995

Link to publication
Design and Realization of a Video-On-Demand Demonstration System

A. Mulder

Master thesis

Eindhoven, October 1995
Coach: Ir. H.P.A. van den Boom
Professor Ir. G.D. Khoe
Preface

Off-course it is almost impossible to work on a project as large as a thesis project without any help of other persons. I want to thank several persons who have been a great help to me, and without whom I could not have finished this project.

First I want to thank my father who has been a great support to me during my thesis period. The project could not have been done without the required hardware, therefore I want to thank Leon Kohlen and Gerk Huizinga for the hardware and the software support of the MPEG-card. Furthermore, I want to thank the people at the 'Rekencentrum' for their support on the compiler and TCP/IP software.

On a less professional base I want to thank my friends and fellow students for their support during the thesis project. You always need friends to relax besides the hard work being done for the project.

And last I want to thank my coach Henry van den Boom, and Professor Khoe for offering me the chance to fulfill my thesis project at the Electro-Optical Communication Group, and all employees working there, who have always been there to help the students.
Abstract

Video-On-Demand is defined as: 'The ability of a person, subscribed to an Information-Provider, to choose a film of choice and be able to watch the film whenever the person wants, using VCR-like functions like pause, stop, fast/slow-forward, fast/slow-backward, and goto to control the playing of the film.'

To provide a Video-On-Demand service a bidirectional network between Information-Providers and subscribers is needed. The Electro-Optical Communication Group performs research on such a network, and research on a Video-On-Demand service.

In this report the design and realization of a Video-On-Demand Demonstration Network is described. This work has been done for the master thesis degree. The Video-On-Demand demonstration system consists of: a Video-On-Demand server application, running on the Windows NT operating system for a PC, and a Set-Top Box application running under DOS. The connection between the server and the set-top box is implemented using Ethernet cards, in combination with TCP/IP software for sending data between the computers. The video information for the system is encoded using MPEG-1. The screens to be shown at the set-top box are implemented as ASCII-files.

The Video-On-Demand server's main functions are: providing the set-top box with a User-Interface for the Video-On-Demand service offering a menu structure for choosing a movie to be watched; handling the subscriber's responses to the menu screens shown at the set-top box; sending MPEG-1 encoded video towards individual set-top boxes when a movie is played.

The set-top box main functions are: decoding and playing the MPEG-1 video information at a television set; sending the users responses of the menu screens towards the server.

The User-Interface is controlled by the server and stored in a linked-list. The linked-list contains: pointers to menu screens to be shown at the subscriber's set-top box; the possible responses to these menu screens; pointers to text screens to be shown at the set-top box; and pointers to MPEG-encoded movies to be sent towards the set-top box. The ASCII-files of the screens to be shown at the set-top box are sent using the Ethernet-link in combination with the TCP-protocol. The messages between the server and the set-top box containing for instance user responses are sent through the Ethernet-link using the UDP-protocol.
Contents

1 Introduction 1

2 General aspects of designing a Video-On-Demand network 3
   2.1 What is Video-On-Demand? .................................. 3
   2.2 The underlying CATV network .................................. 4
   2.3 Adaptation of the CATV network for VOD ......................... 5
   2.4 The VOD server ................................................ 6
   2.5 Set-Top Box for VOD at subscriber’s site ...................... 7
   2.6 Implementation of VOD network ................................ 9

3 The MPEG compression 11
   3.1 Analogue video ............................................. 11
   3.2 Digitizing analogue video ................................... 11
   3.3 The MPEG-1 standard ....................................... 12
      3.3.1 The audio stream ...................................... 12
      3.3.2 The video stream ...................................... 12
      3.3.3 Discrete Cosine Transform, Run-Length Coding and Variable-Length Coding ........................................ 14
      3.3.4 Predictive Coding and Motion Compensation and Estimation ........................................ 14
      3.3.5 Video Trick Modes ...................................... 14
   3.4 Differences between MPEG-1 and MPEG-2 ...................... 15

4 Implementation of the Set-Top box 17
   4.1 Hardware and software setup .................................. 17
   4.2 The TCP/IP software for the set-top box ...................... 18
      4.2.1 TCP protocol ........................................... 19
      4.2.2 UDP protocol ........................................... 20
   4.3 The MPYGMALION software ................................... 20
      4.3.1 The adaptation of the software ......................... 20
      4.3.2 How the Mpygmalion software should be used .......... 22
   4.4 Properties of the set-top box ................................ 23
   4.5 Working of the set-top box software ......................... 24
      4.5.1 Structure of screen-files sent from the server to the set-top box ........................................ 26
      4.5.2 The message structure .................................. 26
      4.5.3 Possible keys the user may use ......................... 27

5 Implementation of the Video-On-Demand Server 29
   5.1 Hardware and software setup .................................. 29
   5.2 The TCP/IP implementation ................................... 30
   5.3 Multitasking ................................................. 30
   5.4 The structure of the User-Interface .......................... 31
   5.5 The working of the server software ......................... 34
5.5.1 The main-thread .................................................. 34
5.5.2 The thread handling the set-top box ........................ 36

6 Conclusions & Recommendations ................................. 39
   6.1 Conclusions ......................................................... 39
   6.2 Recommendations ................................................ 40
       6.2.1 Recommendations for enhancing the set-top box .......... 40
       6.2.2 Recommendations for enhancing the server ............... 40
       6.2.3 General recommendations .................................. 41

A The structure of the file used to build the structure-linked-list 43

B The address of ELKA Automatisering 45

Bibliography 47
List of Figures

2.1 Cable Network Structure ................................................. 4
2.2 The frequency band configuration .................................. 5
2.3 The basic layout of a VOD service .................................. 5

3.1 MPEG video datastream structure .................................. 13
3.2 MPEG video datastream structure .................................. 14
3.3 Comparison of MPEG-1 and MPEG-2 quality ...................... 16

4.1 Block scheme of the set-top box software ......................... 24

5.1 User-Interface example, implemented in this project .......... 32
5.2 Linked-list of example User-Interface ............................ 33
5.3 Main-thread of server application ................................. 35
5.4 Client-Process of server application ............................. 37

6.1 Example of dedicated hardware structure ......................... 41
Chapter 1

Introduction

Throughout history entertainment has been very important to people. Theatres, music and movies, to name a few, have made life more enjoyable. The invention of television added a completely new dimension to entertainment. When at first people had to leave their houses to find entertainment, television made it possible to be entertained at home!

With the invention of television people became accustomed to being entertained at home, being able to see music performances, watch shows, discover other worlds and of course, see movies at home.

When people want to see movies, one possibility is to subscribe to an Information-Provider, who distributes these movies through a cable network to the subscriber’s home, or rent the movies and play them with a VCR. Because of the nature of people, new services keep being developed to make life easier. One of these luxuries is the possibility to choose the movie you want to watch at your home, making it superfluous to even leave your home. This service is called Video-On-Demand.

Video-On-Demand is a service provided by Information-Providers to offer subscribers a choice of movies, and control the playback of the movie with VCR-like functions, like play, stop, pause, fast-forward, etc. Much research all over the world is being done at developing a Video-On-Demand system, especially concerning the problems when a Video-On-Demand service has to be offered to a large group of subscribers.

Naturally, the Optical Communications Group of the Department of Electrical Engineering of the Eindhoven University of Technology is one of those research groups that work on the Video-On-Demand technology. Research is being done at the communication network between the End-Users, the subscribers, and the Information-Providers, the Cable Companies.

One of the goals of the Optical Communication Group is to create a demonstration model of a Video-On-Demand network. This demonstration network can be used for public relation purposes when visitors visit the department, but the main function of the network is to form a test bed for further developing new technologies in optical transmission.

Contents of this report

This report describes the work what was done for a Master of Science project. The work involved the development of a Video-On-Demand network. In the following chapters the results of this work are treated. In chapter 2 the aspects of a Video-On-Demand network are treated. Chapter 3 explains the MPEG-standard used in this project for compressing digital video. Chapter 4 handles the implementation of the set-top box in more detail, while chapter 5 handles the implementation of the server in more detail. Because the Video-On-Demand project is a large project, the last chapter contains some recommendations of tasks that could be performed to further enhance the Video-On-Demand-network. Chapter 6, with conclusions, ends this report.
Chapter 2

General aspects of designing a Video-On-Demand network

This chapter describes several aspects in designing a Video-On-Demand network. First the term Video-On-Demand (VOD) itself will be handled, after which several parts of a Video-On-Demand network will be dealt with, like the underlying CATV network, the necessary network adaptation for VOD, the VOD server and set-top box.

2.1 What is Video-On-Demand?

Video-On-Demand can be defined in the following way:

The ability of a person, subscribed to an Information-Provider, to choose a film of choice and be able to watch the film whenever the person wants, using VCR-like functions like pause, stop, fast/slow-forward, fast/slow-backward, and goto to control the playing of the film.

Video-On-Demand is a service offered by Information-Providers. At the moment, much experimenting with VOD networks is going on, because there is still no network that is economically feasible for use within a large area, where many subscribers want to use the VOD service. Costs of equipment are just too high for a VOD system to be profitable, and underlying networks and protocols to transport the VOD information are still being designed.

In the literature there are several definitions pertaining to Video-On-Demand [Fort93,Wilt93a]. These can be divided in the following groups:

- **Pay-per-View.** In a pay-per-view system the user has the opportunity to choose a movie from a list of movies offered by an Information-Provider. Once a choice has been made, the movie is broadcasted to the user, but no VCR-like functions are offered to the user.

- **Near Video-On-Demand.** In a near Video-On-Demand system, the Information-Provider broadcasts a movie at multiple channels with a time-difference between the channels. This gives the subscribers simple playback control like pausing, by continuing playback at relatively the same position in one of the channels. With this form of VOD many clients can watch the same movie, but problems arise when many subscribers want to watch different movies.
• **Interactive (or Full) Video-On-Demand.** This is the most flexible form of VOD. The user has total control of the playback and choice of the movie. The user can control the playback of the movie with VCR-functions like start, stop, pause, fast forward/reverse, slow forward, etc. This form of VOD implies individual channels for each subscriber.

Many experiments with VOD systems based on one of the methods mentioned above have been done [Wilt93a, Wilt93b, Nade92, Sell92, Nish90, Fort93]. The most difficult system to implement of course is the Interactive VOD system. The other two methods are used to dodge the general problems of Full VOD, and as cost saving interim solutions.

### 2.2 The underlying CATV network

When an Interactive Video-On-Demand system is considered, it is obvious that somehow the subscriber has to be able to communicate with the Information-Provider to make choices pertaining to the service offered. This means that a communication channel from the subscriber to the Information-Provider is needed.

In the Netherlands more than 95 percent of the households have cable television and radio. The present cable networks in the country distribute approximately an average of 22 TV channels each. These channels are broadcasted by Information Providers and distributed by the cable network companies. The structure of the underlying network is more or less hierarchic (see figure 2.1).

![Figure 2.1: Cable Network Structure](image)

The highest level in the network hierarchy is formed by the head-ends. The head-ends can receive information through antennas, satellite dishes, and/or can be connected through high speed trunk-lines. All high-speed trunk-lines are fiber. The head-ends distribute the information to district-stations. After the distribution of the information towards the district-station, the information is further distributed towards starpoints. The last step in the network is the distribution from the starpoints to the subscriber's home. At the moment, throughout the country, the part of the network from the district-station towards the subscriber is formed by coaxial cable. The question remains if it will be feasible to replace these coaxial cables by fiber.

At this moment the Electro-Optical Communication Group of the Eindhoven University of Technology has several projects running which pertain to the problems of designing a bidirectional network suitable to implement a Video-On-Demand service. Work is being done in further describing the properties of the CATV network, designing an upward- and downward-channel for communications between the subscriber and the server and designing the downward-channel for
sending video information towards the subscriber. At the moment, no solid definition has been made for the datarates of the up/down-channels and the downward-channel, but the datarates will approximately be 64kb/s for the up/down-channel, and 2.0Mb/s for the downward-channel.

The network described earlier contains unidirectional amplifiers, which cause problems for the upward-channel from the subscriber to the Information-Provider. Another problem lies in the limitation of the free frequency bandwidth that is to be used for the VOD service. The subscriber probably won’t be very pleased when channels they receive have to be removed for an implementation of the VOD service. Therefore, other parts of the band have to be used. The Electro-Optical Communications Group made the choice of reserving the frequencies from 5-30 MHz and 47-68 MHz for up- and downward communication channels, from subscriber to Information-Provider, and 250-450 MHz for downward-channels from Information-Providers to subscriber (see figure 2.2).

![Figure 2.2: The frequency band configuration](image)

### 2.3 Adaptation of the CATV network for VOD

A Video-On-Demand network is an integrated part of the CATV network. It consists of several parts. First there is a server that offers the information to subscribers. This server, or several servers to supply many subscribers with VOD, could be located at the head-end or district-stations.

One big difference between an ordinary CATV network and a Video-On-Demand network is that a (virtual) bidirectional path between the subscriber and the Information-Provider is necessary. The path is a requirement because when Interactive VOD is provided, the information should be sent towards one subscriber only, and thus be routed. This means that every subscriber needs a (virtual) path to receive its information. In figure 2.3 a basic layout of a VOD service is given.

![Figure 2.3: The basic layout of a VOD service](image)
The (virtual) path between the Information-Provider and the subscriber is used to send the information in digital format. This is in contrast with the present CATV channels, which are analogue signals when received at the subscriber's home. The command messages between the server and the subscriber are digital, but the movie information is also sent digitised. To send movies in digital format to the subscribers, the MPEG standard is used to compress the bulk of the digital video. There are several MPEG compression techniques, but only MPEG-1 and MPEG-2 will be handled here. With MPEG-1 compressing digital video, a datastream with a datarate of approximately 1.5Mb/s is achieved. For a more detailed explanation of MPEG, see Chapter 3.

Then of course the information transported from the server should be handled at the subscriber's home. At the subscribers home a set-top box is located for this purpose. At the moment this set-top box is most often looked at as a true standalone box, but in the future this set-top box should be integrated in the television set.

2.4 The VOD server

The Information-Providers provide the subscribers with the VOD service. For this service a network has to be established consisting of high-speed, large capacity servers for serving many subscribers. These servers can be described as existing of several parts:

- **Storage capacity.** The server should have a storage capacity for storing films (MPEG encoded) and other data. The other data can be data necessary for offering a user-interface to subscribers, and storage capacity used for transport control between the server and each subscriber. The storage capacity could, for instance, exist of harddisks, CD-ROM disks, or RAM.

- **Network-Interface.** The Network-Interface takes care of the connection between the set-top box and the server, taking care of communication protocols, routing, et cetera. Part of the Network-Interface is maintaining a bidirectional up/down-channel and a unidirectional down-channel with each subscriber.

- **Management.** This part of the server takes care of general management-functions, like account-management (payment of the service), database-management and session-management for each subscriber.

- **VCR-functions.** Probably the most difficult parts to realize are the VCR-functions the server for an interactive VOD system should offer the subscribers. The VCR-functions to control the playback of the movies are **play, stop, pause, fast-forward/backward, slow-forward/backward** and **goto**.

An average movie about 90 minutes long, which is compressed according to the MPEG-1 standard, has an average size of approximately 1GByte. If storage at the server takes place with harddisks, for instance RAIDs (see [Drap94]), and many movies are offered to the subscribers, the amount of data gets out of hand. It is therefore a better strategy to distribute the movies over different servers. When this strategy is used, servers should be able to offer movies fetched from other servers. The Network-Interface of the server should take care of accessing other servers for movies and offer movies to other servers.

MPEG-compressed files are quite large files to handle. When a subscriber has chosen to watch a movie, the MPEG-data should be transported towards the subscriber, due to the unidirectional nature of the downward-channel. This should happen with at least the sustained datarate necessary for the set-top box to decode the MPEG-data. The average datarate for decoding the MPEG-data is approximately 192kBytes/s.
There are several methods with which the server can offer the stored MPEG-data to the subscriber's set-top box:

- The server has the complete movie (MPEG-data) available for transport. With this method the server is able to read data from the complete MPEG movie-file and transport the data towards the subscriber.
- The server uses buffers to send the MPEG-data towards the subscriber, so only part of the movie (with the size of the buffer) has to be available at the server at one time. To maintain a dataflow towards the subscriber, the buffer used to send towards the subscriber should be flow-controlled to prevent underflow.

The question arises which of these methods is most efficient. This depends on some different factors. First the number of subscribers the server should be able to serve is a decisive factor. When a small network is considered, for instance a local network within a hotel, a server with all the movies on local storage capacity is most cost-efficient because of the limited amount of movies to be offered for a hotel environment, because it is flexible, and because the number of users at one time to view a movie is limited. MPEG-data to be used by the set-top boxes can be read directly from the local storage capacity, for instance harddisks.

In larger networks used by many subscribers, transport problems arise when much data should be offered to subscribers, because the datarates at trunk-levels are much higher than at end-user-level. Therefore buffers are needed to take care of datarate fluctuations, and because data switching should take place. Data switching becomes an important factor when data to be offered to subscribers come from servers located at other parts of the network.

2.5 Set-Top Box for VOD at subscriber's site

Each subscriber has a set-top box (STB) at home for connecting with a network that can deliver VOD service. The STB should exist of several parts:

- **Cable TV.** The reception of standard TV channels the subscriber already receives cannot be altered. The set-top box should therefore be an extension of the cable, enabling normal reception of TV channels.
- **Network-Interface.** The network-interface connects the STB with the network offering the VOD-service. For each subscriber in a VOD network there should be a (unidirectional) high-speed downward channel (at least 1.5Mb/s for sending a sustained MPEG-1 datastream) towards the STB, and a bidirectional up/down channel (app. 64kb/s) for sending control data between the server and the STB.
- **MPEG Decoder.** The MPEG-1 video-data should be decoded at the STB and be made visible on a standard television set. The STB therefore needs hardware to guarantee real-time decoding of the MPEG-1 information. Hardware that decodes MPEG-data reads data at a rate of approximately 1.5Mb/s
- **User-Interface.** The STB should contain a user-interface to offer at least the VCR-functionality's the subscriber wants when using a VOD-service. Besides the VCR-functions offered, there should also be a proper Man-Machine-Interface (MMI), preferably consisting of a clear graphical menu-structure. This way, several options could be offered to the subscriber, i.e. a choice of services, search-facilities for choosing films and obtaining personal management data.

The User-Interface

The User-Interface (UI) is an important aspect of the Video-On-Demand system, because it determines if subscribers are willing to use the VOD-service or not. Most important is the user
friendliness of the service. Many different User-Interfaces have been implemented and tested (see [Sel92, Hodg93, Wilt93a, Wilt93b]), like using a (computer) terminal for selection purposes, Hyper-Text-Macro-Language for creating a WWW-like environment, On-Screen-Display, et cetera. The user-friendliest is probably a system that uses a menu structure through which the user can navigate. This can be either a graphical menu structure, or for instance a textual menu structure for On-Screen-Display.

There are several strategies for implementing a User-Interface structure:

1. The *intelligence* is placed at the set-top box. This strategy has the User-Interface implemented on the set-top box, i.e. the menu structure, existing of a graphical or textual layout or another User-Interface, is stored at the set-top box. The server is used as a storage capacity, for routing, and for subscriber-management. The STB has to keep track of the users-actions and handle these actions.

2. The *intelligence* is placed at the server. This strategy has the menu structure forming the User-Interface, located at the server. This means that parts of the menu structure, for instance background-screens, should be transported towards the set-top box. The server keeps track of the subscriber’s actions and reacts on these actions.

ad 1 The disadvantages of this strategy are the complexity of the STB that needs storage capacity for the (complete) User-Interface. Furthermore, the system is not very flexible in the sense that changes in the User-Interface should take place at all households that use the system. The advantage is the relative simplicity of the server concerning the users-actions, whose main task is transporting data towards the subscriber.

ad 2 The disadvantage of this strategy is the necessity of a more complex server that keeps track of the users-actions, thus needing more computational power and speed. Advantages are a cheaper STB that needs no (or few) storage capacity, and which is flexible in the sense that changes in the User-Interface immediately affect all the subscribers households.

The VCR-functions

Another important aspect of the Video-On-Demand system is the VCR-functions the subscriber has to have to get a full interactive VOD-system. The VCR-functions are those functions that pertain to the playback of the MPEG-decoded movie. These functions influence the reading of MPEG-data and the transport of the MPEG-data towards the subscribers. See Section 3.3.5 for a more detailed explanation of the VCR-functions available for MPEG-decoded video.

There are several strategies for implementing the VCR-functions:

1. **Server.** The VCR-functions are implemented at the server. The server adapts the MPEG-dataflow towards the subscriber when the subscriber uses VCR-functions like fast-forward/reverse.

2. **Set-top box.** The VCR-functions are implemented at the set-top box. The VCR-functions now affect the decoding of the MPEG-data at the STB. The server has to ensure proper arrival of MPEG-data at the STB.

3. **At server and STB.** The VCR-functions are now implemented at both the server and the set-top box and are a combination of the methods mentioned above. The set-top box reacts to the users-actions of VCR-functions, while the server also affects the transport of the MPEG-data coinciding with the requested VCR-function.

ad 1 With this method the subscriber’s commands are transported towards the server through the upward-channel. The server acts on these commands and affects the sending of the MPEG-data, i.e. when the subscriber wants to use the fast-forward function, the server has to alter the transport accordingly (see Section 3.3.5).

ad 2 With this method, a buffer is needed at the set-top box to process the MPEG-data for the VCR-functions. The reason for the buffer is that for certain VCR-functions data within the
MPEG-stream has to be skipped, but the datarate for decoding the actual MPEG-data has to be obtained (1.5Mb/s).

ad 3 With this method, the set-top box responds quickly to the users-actions, and meanwhile sends the commands also to the server. The server than handles the commands to implement the VCR-functions.

2.6 Implementation of VOD network

There is a difference between a VOD network suitable for commercial purposes and a VOD network for demonstration and experimental design purposes. The goal of the VOD project of the university is not to put a VOD system on the market, but to perform research at certain aspects of VOD, most of all the underlying networks, and to have a demonstration VOD-network. Therefore a VOD system should be developed which gradually grows towards a larger system with a growing amount of subscribers. The first goal is to create a VOD-system for just one or two subscribers, built so that it is relatively easy to serve more subscribers by adding hardware. Emphasis will be put on the basic structure of the system. When this basic structure is laid down, it can gradually be expanded.

When one looks at a VOD system that should eventually be available for cable companies, clearly this system exists of dedicated hardware (and software) to obtain the datarates necessary for serving many subscribers distributed over a large area. This dedicated hardware is especially important for the network-routing and transport, because for instance a system suitable for 1000 subscribers might at one point, when all subscribers want to watch a movie, or worse, the same movie, need to transport 1000 1.5Mb/s datastreams. All over the world much time and money is spent to develop such a system.

For this project the aims and the resources are different. The demonstration network will therefore be a simulation of a real-time dedicated hardware oriented VOD system. Instead of using dedicated hardware to obtain a hardware oriented connection between the server and the set-top box, the connection between the server and the set-top box will be implemented with an Ethernet-link. To send data across an Ethernet-link, several protocols can be used depending on the computer platform (UNIX, DOS). Two of such protocols suitable for use in this project are NETBEUI (for DOS), and TCP/IP (platform independent). The first protocol can be used to transport files from one location at the harddisk of any PC, towards an other location at any PC's harddisk connected through an Ethernet-link. The second protocol, TCP/IP, can be used to transport arbitrary data across an Ethernet network, thus transport parts of files or other user-defined data.

This leads to two possible choices of transport methods for providing the set-top box with MPEG-video.

File Transfer

The first method uses a NETBEUI protocol between two computers. This protocol enables two (or more) computers that are interconnected through Ethernet-cards to share files with each other. This way the computer that serves as the STB can read files from the harddisk of a computer that serves as the server. These files can then be shared by several STBs connected to the server. The STB can read directly from the hard-disk at the server, or can copy the file from the server to a local storage-capacity at the STB. When this method is used, the server forms a passive storage capacity, while the User-Interface is handled at the set-top box. Thus all the intelligence of the VOD-service is then located at the subscribers STB.

This method means whole MPEG-files containing the movie should be present at the server or should be accessible through another server. If the complete file is copied to a local disk at the
2.6. IMPLEMENTATION OF VOD NETWORK

STB, a large storage capacity is needed because MPEG encoded movies are considerably large. Therefore, sharing files from the servers harddisk and just read from those files is a better option.

When NETBEUI is used for sharing files, deviation of the VOD-network has taken place, because the data transport takes place through a bidirectional Ethernet-link, and because the set-top box is the one controlling the transport, while the intended CATV-network comprises an unidirectional downward-channel.

An up/down-channel is needed to send command-messages between the server and the STB. These commands are used to initiate a VOD-session, perform subscriber-management, perform database management, send subscriber responses (user-interface) to the server, etc.

Buffered streams

Another way to build a software-based demonstration-network is by using buffers. There are two logical ways to use buffers. These are:

- **Buffer at the server.** Here a buffer for each subscriber is placed at the server. The buffer located at the server should be filled with the proper MPEG-information, and be sent towards the set-top box. The set-top box now has the task to handle the transported data accordingly. When during a VOD-session the subscriber wants to play a movie or do tricks-states (fast-forward, goto, etc.), the server should keep the buffer filled with the proper MPEG-video-information to be sent to the STB.

  For the connection between the server and the STB an Ethernet-link can be used in combination with an existing protocol like TCP/IP. The advantages of this protocol are that TCP/IP guarantees successful transmission and is a widely accepted standard.

  An up/down-channel is needed to send command-messages between the server and the STB. These commands are used to initiate a VOD-session, perform subscriber-management, perform database-management, send subscriber responses (user-interface) to the server, etc.

- **Buffer at server and STB.** Another structure using buffers is by placing a buffer at the server and one at the STB. The buffer at the server has to be (continuously) filled with the proper MPEG-video-information and should keep the buffer at the set-top box filled accordingly. When during a session between the STB and the server a video has to be played, the server sends MPEG-information towards the buffer at the STB and the MPEG-decoder unit reads from this buffer. With this setup, when the subscriber chooses to play video, the server should send the MPEG-information towards the STB, but the STB should synchronize this dataflow, to avoid under- or overflow of the buffer at the STB. At both the server and the STB there should be a mechanism that keeps the buffers filled. This calls for synchronization of the down-channel communication between the server and the STB.

  With this method the tasks of the STB become more complex, because mechanisms are needed to keep the buffers at the server and the STB filled. For the connection between the server and the STB an Ethernet-link can be used in combination with an existing protocol like TCP/IP.

  An up/down-channel is needed to send command-messages between the server and the STB. These commands are used to initiate a VOD-session, perform subscriber-management, perform database-management, send subscriber responses (user-interface) to the server, etc.
Chapter 3
The MPEG compression

3.1 Analogue video

An analogue video-signal is defined by the number of frames per second and the number of lines per frame. The number of frames for the PAL-standard, which is used in the Netherlands, is 25 per second, and the number of lines is 625 per second. Each frame is built up with two fields (odd and even field), each field contains half the frame information and is shown at a 50 Hz rate. The odd field contains the odd line-numbers of a frame, while the even field contains the even line-numbers of a frame. The fields are then shown interlaced. Interlaced means that first the odd lines are shown on the screen, then the even lines are shown. Because of the structure of the monitor tube, the shown lines slowly fade away, thus when a new field is shown the previous field is still visible leaving the human eyes the impression of a complete picture (frame). Video-information exists of several components, a luminance (Y) signal with the black-and-white video-information, and two colour difference signals (Y-B and Y-R, or respectively U and V). The video-signal also contains blanking periods for horizontal and vertical synchronisation information, so not all the information is active video.

3.2 Digitizing analogue video

Analogue video is digitized according to the CCIR-601 standard. Because analogue video contains redundant information for synchronisation purposes, less lines per frame are obtained with active video. Therefore, only 576 lines per frame are left to be digitized. The active video (Y signal) in the lines is sampled with a sample rate of 13.5 MHz. This creates 720 pixels per line, having 10 bits per pixel. The colour difference signals U and V are sampled at half the luminance sampling rate, 6.75 MHz. With these sampling rates the following bit-rate for digitized video is obtained:

\[
\text{Luminance } Y : \quad 720 \times 576 \times 25 \times 10 = 103,680,000 \text{ bits per second}
\]
\[
\text{Chrominance } U : \quad 360 \times 576 \times 25 \times 10 = 51,840,000 \text{ bits per second}
\]
\[
\text{Chrominance } V : \quad 360 \times 576 \times 25 \times 10 = 51,840,000 \text{ bits per second}
\]

Which leaves a total bitrate of 207.36 Mbits per second for the digitized video-signal, using 10 bits per sample. Apart from the video-information, there is more data added for ancillary data (for instance for teletext), and for up to 4 audio channels. This leads to a total bitrate of 270 Mb/s for the complete video-signal according to the CCIR-601 standard.

This bitrate is too high to be of practical use for transport over a large network, because of limitations in distribution channels. Digitized video is therefore only used when the digitized video has to be edited, i.e. in TV-studios. Compression techniques could be used to reduce the bitrate...
of digital video to make it more manageable. The International Standards Organisation (ISO) Motion Picture Experts Group (MPEG) is an institute that defined standards for compression algorithms: The MPEG-standard, like MPEG-1, MPEG-2 and MPEG-4. The MPEG standards specify the format of the bitstream for the encoded video so that it is relatively straightforward to design a proper decoder.

### 3.3 The MPEG-1 standard

The MPEG-1 standard, also known as the ISO11172 standard, is a flexible standard that describes the compressing and decompressing of motion pictures (video) and audio, and the multiplexing of the video, audio and ancillary service data. The MPEG-1 coding scheme comprises of several layers. The **system layer** contains timing and other information for demultiplexing the video and audio, and a **compression layer** which contains the compressed video and audio. In the following subsections a summary of the MPEG-1 datastream will be given. For a more detailed description of the MPEG-1 standard, see [1191].

#### 3.3.1 The audio stream

The MPEG datastream exists of two compressed audio signals of 224 kbit/s, thus having two audio signals of 112 kbit/s each. The audio signals can be compressed by sampling the analog signals with either 32, 44.1 or 48 kbit/s. In practice, only 44.1kb/s is used to sample audio for movies, according to the CD-standard. The two compressed datastreams are multiplexed into the MPEG datastream.

#### 3.3.2 The video stream

Although the MPEG-1 standard is designed to specify a wide range of datarates from 1 to 10Mbit/s, the MPEG-1 standard is tuned to perform well at a datarate between 1.0 and 1.5 Mb/s, with spatial resolutions of approximately 350 horizontal pixels and 250 vertical pixels (spatial resolutions with a maximum of 4096 by 4096 pixels are allowed) and a framerate of about 25 to 30 frames per second (a maximum of 60 is allowed). These resolutions are chosen to coincide with the structure of standard digitized video based on the CCIR-601 standard. The MPEG-1 standard codes progressively scanned pictures (pictures that contain all the scan lines) and does not recognize the concept of interlace, where a picture is built up by two fields, each containing half the scan lines. In the following subsection the compression techniques for the video, and the structure of the videostream are shortly dealt with.

**Picture format for MPEG-1**

The first reduction made in encoding the videodata is the reduction of the picture size. From the video defined by the CCIR-601 standard, only the odd-field containing half the frame (picture) data is used. This leaves 720*288 pixels by lines. After clipping the lines, 704 line pixels are obtained, after which a filter is used to reduce the horizontal resolutions to obtain a 352*288 pixels by line picture format. This format is called the **SIF format**.

**The video datastream structure**

The smallest defined structure in the video datastream is a **block**. A block consists of 8 * 8 pixels array values, representing the luminance and chrominance video information. See figure 3.1.

Several blocks are combined in a **macroblock**. A macroblock is used to represent the coded pixel data. One or more adjoining horizontal macroblocks are grouped together to form a **slice**. Slices are used to handle errors during decoding of the MPEG information. If an error occurs
in the datastream, the decoder can skip data till the next slice is reached. A group of slices is combined to form a frame or picture. The picture consisting of several slices make up the active picture area visible on the screen. One or more pictures are combined to form a Group Of Pictures (GOP). The GOP is used for random access in the datastream.

The MPEG-standard defines three kinds of pictures. These are:

- **I-pictures.** This stands for Intra-coded pictures. I-pictures are used to represent the actual picture, independent of former or future pictures. They make random access in the video datastream possible, and are used to handle several trick-states (see Section 3.3.5). The I-pictures use three methods for compressing the digital picture. These are discrete cosine transform (DCT), quantization and run-length coding (see following subsection).

- **P-pictures.** Predicted pictures. These pictures are predicted pictures based on the closest former I- or P-picture. This is called forward prediction, and uses Motion Estimation and Compensation. P-pictures provide more compression than I-pictures, but propagate coding errors when P-pictures (or B-pictures) use prior P-pictures to predict on.

- **B-pictures.** Bidirectional-coded pictures. These pictures use both future and past pictures (I- or P-pictures) to predict a picture from. This picture provides the most compression of the three types. They do not propagate errors because they are never used as a reference picture.

The MPEG standard allows the system designer to determine the sequence of I-, P- and B-pictures in the video datastream. In figure 3.2 a typical display order of pictures is given. The display order does not have to be the same as the bitstream order, because B-pictures also use future pictures. This means buffers have to be used to encode and decode the B-pictures.

The number of I-pictures determines the random access possibilities of the video stream. The more I-pictures, the better random access is. But I-pictures provide less compression than P- or B-pictures, thus more I-pictures mean a higher datarate. On the other hand, more I-pictures mean a better quality of the MPEG-video. The system designer therefore has to make a tradeoff between random-accessibility, video quality and datarate.
3.3. THE MPEG-1 STANDARD

Display Order

IBBBBPPBBBPBBP

1 2 3 4 5 6 7 8 9 10 11 12 13

Bitstream Order

IPBBBPPBBBPBBB

1 4 2 3 7 5 6 10 8 9 13 11 12

Figure 3.2: MPEG video datastream structure

3.3.3 Discrete Cosine Transform, Run-Length Coding and Variable-Length Coding

A block consisting of 8 by 8 pixel values has a high spatial redundancy. This redundancy can be removed without causing too much loss of quality for the human eye. The reduction is done in several steps. First, DCT is applied. This is done by transforming a block to the frequency domain. The result of this is an array of the same size containing the coefficients of the frequency components. Because of the spatial redundancy, many higher frequency components are zero.

Compression is now achieved by quantization of the frequency component, thus limiting the possible values. This implies an irreversible loss of information.

Further compression is achieved by combining the quantized frequency components in one sequence, and using Run-Length Coding (RLC). With this coding technique, pairs of non-zero and zero components are formed and given a certain codeword, which on average causes fewer data because the RLC codewords are smaller than the number of zeros.

The last compression step is achieved by using Variable Length Coding. VLC uses shorter codewords for frequently occurring RLC-pairs, and longer codewords for less frequently occurring RLC-pairs (Huffman coding). See [Wood93,Baro94,a1.91] for more information on DCT, RLC and VLC.

3.3.4 Predictive Coding and Motion Compensation and Estimation

The P- and B-pictures make use of predictive coding and motion compensation/estimation. In the MPEG encoder, motion estimation is used, while in the decoder motion compensation is used. Predictive coding is a technique to achieve compression by removing temporal redundancy and/or quantization. This technique, based on pixels previously decoded, predicts the value of a pixel for the next picture. This results in a difference between the true value and the predicted value. The difference can now be quantized and encoded, leading to a higher compression than when true values are encoded.

P- and B-pictures use motion compensation to exploit temporal redundancy in the video. Motion Compensation is a method to estimate or predict the value of a block or macroblock in the picture to be decoded, by using a two-dimensional motion-vector indication the shift of a previous block or macroblock to a new location, because of motion. It is usually more efficient to transmit the motion-vector in combination with the difference between the predicted block and the current block, than to transmit a description of the current block itself.

3.3.5 Video Trick Modes

The MPEG standard is designed to provide several VCR-like functions in displaying the MPEG-video data. These are: forward play, freeze picture, fast forward, fast reverse, slow forward and random access. The functions, except forward play, are known as trick modes.
The properties of the trick modes depend on the implementation of the MPEG-stream, especially the number of I-frames used and the sequence of the I-, P- and B-pictures. The speed of the fast forward (or fast reverse) playing mode is determined by the number of I-pictures in the stream, because when this mode is required, only the I-pictures are decoded and shown. If there are I-pictures at intervals of 10 pictures, fast forward/reverse can play 10 times as fast. This trick mode places a burden on the media and the decoder, because the datarate of the input datastream should be increased 10 times, and the decoder should be able to decode this at a higher datarate.

There is another way of implementing this trick mode. It is also possible to achieve fast forward/reverse with a constant bitrate. This is achieved when the medium transporting the MPEG-datastream sorts out the I-pictures, and only sends the I-pictures at the normal datarate, skipping the P- and B-pictures in the datastream.

It is also important to realize that when encoding the MPEG-1 datastream, the sustained datarate should be obtained. That means that when there is few information in the video, bitstuffing takes place to reach the datastream. Another solution also used instead of bitstuffing is to let the datastream only exist of I-pictures, because I-pictures require more data to be encoded.

### 3.4 Differences between MPEG-1 and MPEG-2

A later standard developed by the Motion Pictures Experts Group is the MPEG-2 standard. There is a lot of confusion about this standard, as it is often thought that this standard is a replacement for the MPEG-1 standard. It is not. The MPEG-2 standard is targeted at higher datarates than MPEG-1, although MPEG-2 can be used at low datarates and MPEG-1 can be used at high datarates. There are however some important differences between the two standards.

The main difference is that MPEG-2 uses interlaced video, and not sequential video as MPEG-1 does (see section 3.1). Thus MPEG-2 incorporates the interlaced structure of analog video, using both the even and odd fields of a frame. This increases the picture quality since the datarate is increased. This leads to another difference between the two standards.

The MPEG-1 stream uses SIF picture sizes (352*288), while the MPEG-2 standard uses full video sizes (720*576). When the datarates of both streams are increased above approximately 3 Mbits/s, the quality of MPEG-1 remains lower than that of MPEG-2 (see [Nels94]). When datarates are lower than 3.5 Mbits/s, the quality of MPEG-1 is higher than that of MPEG-2. This is why MPEG-2 is more suitable at higher datarates, leading to better quality of video. MPEG-1 should be used at datarates below app. 3.5Mbit/s, and MPEG-2 should be used at datarates of about 5.0Mbit/s. This leaves a 'no mans land' between 3.5 and 5.0 Mbit/s. This is shown in figure 3.3. In this figure the quality of the video as shown for MPEG-1 and MPEG-2. See also [Nels94].

Furthermore it is very important to realize that MPEG-2 is has a flexible datarate. This is especially important when MPEG-data is read from CD-ROM. According to the new MPEG-2 standard for multimedia CDs, the rotating speed of the CD-ROM depends on the information in the MPEG-datastream. When much information is present in the video sequence, the speed of the CD-ROM drive is high. When few information is present in the video sequence, the speed of the CD-ROM is low.

Encoding and decoding MPEG-2 is costlier than MPEG-1, but at higher datarates the quality is much better. At a datarate of about 6.0 Mbit/s, standard quality video is obtained, while at a datarate of 10 Mbit/s studio video quality is obtained. For HDTV, 30 Mbit/s is needed.
3.4. DIFFERENCES BETWEEN MPEG-1 AND MPEG-2

Figure 3.3: Comparison of MPEG-1 and MPEG-2 quality
Chapter 4

Implementation of the Set-Top box

In this chapter the implementation of the set-top box is described. First the hardware and software setup of the set-top box is described. Then the adaptation and use of secondarily manufactured software are described in more detail, after which the structure of the set-top box software is dealt with.

4.1 Hardware and software setup

Hardware setup

There are several platforms on which a set-top box can be implemented. For instance: A PC-platform, a UNIX-platform, a MacIntosh-platform, dedicated hardware, etc. Chosen to use is a PC-platform to implement the set-top box. There are several practical reasons for this.

First, when a set-top box is developed, the main purpose of this set-top box is to display video-information, offered by the Information-Provider by means of a VOD server. Because the video is transformed to digital format and is compressed according to the MPEG-1 standard, the set-top box should contain a decoding part for regaining the video information. There are both software and hardware decompressing methods. The software method implies a platform that is capable of real-time decompressing MPEG-1 code. This means the platform on which the set-top box is based should have sufficient computational power. When using the hardware method, the real-time video decoding is handled by the dedicated hardware, leaving processor time to other processes for instance maintaining the Man-Machine-Interface between the user and the offered service, and the connection between the VOD server and the set-top box. The obtainability of hardware decoder cards for the PC-platform is the main reason the PC-platform has been chosen.

Other reasons for choosing the PC-platform are the fact that much hardware and software for the PC is easily obtainable, the prices for software and hardware for a PC are lower than for other platforms, and it is in general easy to use and 'to get into'.

There are several manufacturers that offer MPEG-decoders. All of them contain software that can let a user view MPEG-encoded video stored in one file on harddisk or CD-ROM. Some manufacturers also offer Software Developers Kits (SDK) to create an own application capable of viewing MPEG-decoded video. But almost all of these SDKs handle files in their software-libraries, and do not offer the chance to create an environment that does not read from a file but from, for instance a buffer. There is one manufacturer 1 however which could add the complete source-code of their

---

1 This manufacturer is 'ELKA Automatisering' based in Amersfoort. The MPEG-card they manufacture is called the MPYGMALION. The card is still under development, but for our purpose it is very suitable. My advice however it to keep contact with the manufacturer to obtain the latest software available.
MPEG-card, thus allowing us to rewrite the code to be used in this project. There is one slight disadvantage about the software of this card: the software runs under DOS, and is not written for use under the Windows environment. This means that the software for the set-top box should be developed for running under the DOS environment.

Software setup

Another part of the set-top box software that has to be chosen, is the software. To implement the connections between the server PC and the set-top box PC, the TCP/IP standard is used. The reason for this is the fact that TCP/IP is a widely used protocol for data transport for which Software Developers Kits (SDK) are easily obtainable. Furthermore, there is much information available on the use of this software to implement data transport. The SDK for the TCP/IP stack used must be accessible under DOS. There are three TCP/IP stacks that offer an SDK for use under DOS that were examined for use in this project:

- **WATTCP** This is a shareware product, obtainable from internet through ftp. It is a collection of source-codes (for the C language) which offer some standard TCP/IP C-routines. The software creates a kernel that simulates a multitasking environment, necessary for running several processes for maintaining the TCP/IP stack.

- **3Com 3+Open**— This is an SDK developed by 3COM, named 3+Open TCP Developer Kit. This kit provides the programmer with a DOS-library containing TCP/IP routines, and uses several Terminate and Stay Ready (TSR) programs for creating the TCP/IP stack used by the library routines.

- **PC-NFS** This is an SDK developed by Novell. It also provides the programmer with a DOS-library, and uses TSR programs to create a TCP/IP stack.

After experimenting with the WATTCP and 3+Open SDK's, the choice was made to use 3Com's 3+Open. The SDK PC-NFS was not tested because this could only be used in combination with the Microsoft C Compiler 6.0 or higher, which was not available at that time. 3Com 3+Open was chosen because during experiments a datarate of approximately 370 kBytes/s was obtained when at the server side a file was continuously written to the client, while the client read at will from the data offered. The WATTCP software didn't offer an easy programming environment to do the same experiment as mentioned above, and it might be unstable when used in combination with the interrupt driven software of the MPEG-card because of the kernel that is also interrupt-driven.

The programming for the set-top box was done using Borland C++ 4.5. This choice is arbitrary, but Borland C++ is used often at the university and offers a nice development environment.

4.2 The TCP/IP software for the set-top box

The TCP/IP software used for the set-top box is the TCP/IP stack from the 3Com company, the 3+Open TCP Software Developing Kit. The software package implements the TCP/IP standards, which consist of several protocols, in the computers operating system. This SDK offers the opportunity to perform data transfer between two computers connected through a network. The SDK is based on the Berkeley Software Distribution (BSD) sockets interface, developed in the early 1980s. This interface defines data structures and system calls that use sockets as a way to transfer data over an Ethernet network between two or more computers in a multi-vendor environment.

The BSD sockets are based on the way the UNIX operating system uses handles for devices. For instance, in UNIX file handles are used to read and write from and to a file. Once a file is opened and given a file handle, the file can be read and written to by means of this file handle. Sockets have a similar background. A socket is a handle that is created with just one system
call. After the socket is created, the properties of the socket can be changed. The socket is a handle, like a file handle, which can be used to read from and write to, according to the TCP/IP standard. Once a socket is created, data can be written to it or read from it, and the TCP/IP protocol handles the way the data is sent, routing the data correctly, breaking larger data blocks into several smaller packets and making sure the packets are put back in order, to retain the original data blocks. The TCP/IP protocol defines two protocols that are suitable to use for the communication between the server and the set-top box. These are:

- **TCP.** This protocol is a connection-oriented protocol. With this protocol two computers can have a reliable, flow-controlled, end-to-end stream service.

- **UDP.** This protocol is a connectionless protocol which provides both pair wise communication between a client and server or many-to-one communication between several clients and a server.

Using the TCP/IP socket interface the intended CATV network consisting of a VOD-server and one or more set-top boxes can be simulated by creating a high-speed downward-channel and a lower-speed up/down-channel. The simulation of the channels follow the constraints of the CATV network. For the downward-channel for instance, a sustained datarate of approximately 2Mb/s should be obtained with a reliable datatransfer. It is important to realize that the dataflow exists of real-time data, and should therefore be sustained. For this channel the TCP protocol can be used. For the up/down-channel it should be possible to send command-messages between the set-top box and the server, at a lower datarate. To implement this channel the UDP protocol is very suitable. In the following paragraphs a summary of the characteristics of the two protocols will be given. For a more detailed description of the TCP/IP protocol and the used socket calls, see [Come91,Stev90,Ramb90].

### 4.2.1 TCP protocol

The TCP protocol has the following characteristics.

- It is a connection-oriented, reliable stream-service between two sockets at two applications, i.e. two computers connected by a network. TCP further verifies that data arrive, and in correct order.

- In the definition of the TCP protocol, TCP guarantees that data is sent properly, but does not specify when and how fast the data is sent. In practice, these values depend on the underlying network and hardware, and the way the applications are written.

- When data is sent there is no way of knowing in advance how much of that data is sent by a single send system call. This means that when for instance 20 kBytes of data has to be sent, several send system calls might be needed to send the data. The application should take proper measures for this. The same property applies to the receive system call. When a receive system call is made, it is unknown in advance how much data will be read. It can be one byte, or the complete data packet at once.

- When a server and a client want to communicate with each other through a TCP connection, the following actions are taken. First, the server creates a socket that listens to incoming calls. When a client wants to connect to the server, the client creates a socket and connects to the server. For this the client needs to know the existence of appropriate servers, and the name and addresses the servers can be reached by. The names and addresses of locally

---

2 See [Ramb90]
connected computers are declared in files. \(^3\) Once the server and the client are connected, the connection stays open and reliable data communication can be performed. When the communication is finished, the connection between the server and the client is closed.

### 4.2.2 UDP protocol

The UDP protocol has the following properties.

- The UDP protocol is connectionless. There is no guarantee about reliable delivery. Client and server applications must take proper measurements to detect and correct such errors. On the other hand, UDP does not introduce errors, but depends on the underlying network for proper delivery. Thus UDP works well when the network works well. When using a local network, UDP is a reliable protocol.

- When UDP is used, the data to be sent is sent in one message. This means only one send system call has to be used to send a message to the server, and only one receive system call has to be made to receive the complete message. The maximum size of the messages is determined by the receive and send buffer size of the TCP/IP protocol.

- When a client and a server want to communicate with each other, no connection between the server and the client has to be made. Instead, the server can create a UDP socket and listen to incoming messages from any client. Once a message from a client is received, the address from the sender is noted to enable the server to send a message back to the client that sent the data.

### 4.3 The MPYGMALION software

#### 4.3.1 The adaptation of the software

With the MPEG-card, the source-code of the software was obtained, a requirement for the MPEG-card to be used in this project. But there was one problem with the software as it was obtained, which was caused by the TCP/IP software form 3Com. The TCP/IP software is written for large memory-model programming, while the MPEG-card software was written for the small memory-model. The difference between small and large memory model is crucial and will be explained in the next paragraph. The difference between these models are a problem when programming is done in assembly language, which part of the software for the MPEG-card is. Therefore, the software has been rewritten.

At the beginning of the PC era, MS-DOS was developed, an operating system of which the memory system was based on the use of segments. The complete memory was built up out of segments of 64 kBytes size. This is a 16-bit operating system, i.e. data and addresses can at most be 16-bits wide. Today, MS-DOS still uses the system of segments, but operating systems like Windows NT, Windows 95 and OS/2 use linear memory-models. These are 32-bit operating systems that use 32-bit sized addresses. On these linear memory-models, each memory space has its individual address which is unique. In the 16-bit operating system, memory space addresses are built up by a 16-bit address for a segment and a 16-bit offset in this segment. Programming in C and assembly offers the chance to choose a memory-model with which the compiler builds the application. The memory models available are: tiny, small, medium, large and huge. These memory models refer to the way data is stored, and the way the code of the program (the

---

\(^3\)The TCP/IP protocol uses several files to lay down the structure of the network. These files can mostly be found in the directory `/SYSTEM/ETC/DRIVES: hosts, services, protocols, network`. The file `hosts` contains the names of the computers in the network, the `services` file defines ports used in combination with a service and a protocol. See [Ramb90] for a more detailed explanation.
application) is stored. Only two models will be described here, the small and large memory model.

The small memory model stores all data and the complete code of the program in one segment. This means that the data + code cannot be larger than 64 kBytes. The compiler now uses one segment to store the data and the code, and uses only offset addresses to access data and code. The large memory-model can use several segments, which means that also segment addresses should be used to have access to the data and the code. These differences only refer to programming in assembly, not to programming in the C-language, because the C-compiler takes care of accessing data and the program code, while when programming in assembly the programmer should take care of accessing the data and the program code from the right segment.

The software of the Mpygmalion MPEG-card exists of several modules (files) containing C-code and one module containing assembly code. These modules are compiled and linked together to form the main program, the Mpygmalion software to view MPEG-encoded video from harddisk or CD-ROM. The original software as mentioned before could not be used in combination with the TCP/IP software because it was written for the small memory model. The TCP/IP stack, consisting of Terminate-and-Stay-Ready (TSR)programs, provide a library for programming using TCP/IP system calls under DOS. The TSR-programs are installed once in the autoexec.bat file, and stay in memory. This library is written for the large memory model. Therefore, the software for the Mpygmalion-card had to be rewritten for the large memory model. This meant that the module containing assembly code had to be rewritten. This was a time-consuming job. There are several important changes made in the assembly part that are mentioned here for future programmers who might use the code. The main changes are summarized.

- The assembly module exists of a part initializing the data, and a part consisting of program code. The program code is separated in a part containing procedures that are called by C-subroutines in other modules and a part containing subroutines that are called by assembly-subroutines in the first part of the assembly-module. The original source-code of the Mpygmalion MPEG-card was written with the Zortec C Compiler. To use the code in the Borland C++ environment, which uses a program called TASM to compile the assembly code, the code had to be changed. Originally, the subroutines were addressed by a label. To work in TASM, these had to be changed into procedures. An example of the change follows.

  - The original subroutines used labels:

    ```
    _MPEG3400init:
    push bp
    mov bp,sp
    mov ax, [bp+4]
    pop bp
    pushad
    call INIT.3400
    popad
    ret
    ```

  - The changed subroutine became:

    ```
    _MPEG3400init PROC FAR
    push bp
    mov bp,sp
    mov ax, [bp+6]
    pop bp
    ```
4.3. **THE MPYGMALION SOFTWARE**

```
pushad
    call INIT_3400
popad
ret
```

- Another difference between the Zortec compiler and the Borland compiler in combination with TASM is the way the interrupt routine for the audio decoding is written. For Borland a new interrupt handler had to be written, saving the old interrupt-vector and setting the new interrupt vector to call the correct audio interrupt-routine when an interrupt is generated by the hardware. For correct working the old interrupt-routine should also be handled, and should therefore be chained in the new interrupt-routine.

- When a C-routine calls an assembly routine and parameters are part of the call, the values of the parameter(s) are stored on stack. When these parameters use pointers to data structures, the pointers consist of addresses. As mentioned before, the structure of addresses differ in large and small model. Large memory-model routines use far pointers, which consist of a segment address and an offset. Small memory-models use near pointers that consist of only the offset. This means that when far pointers are written to stack they occupy more space (twice as much, 32-bits instead of 16-bits) than near pointers. To take care of correct parameter passing, the original assembly routines had to be adapted.

An important change in the software of the Mpygmalion-card is the fact that the reading of the MPEG-data had to be changed. The original software has two options for obtaining MPEG-data: reading MPEG-data from a file and reading MPEG-data from CD-ROM. This had to be changed to make it possible to read the MPEG-data from the downward-channel, i.e. the socket with which this downward-channel is implemented. For this, changes have been made in software routines used for reading MPEG-data from CD-ROM. This means that when the software of the Mpygmalion-card is used for the set-top box, the software thinks it reads the MPEG-data from CD-ROM, but instead of using the routines for getting the data from CD-ROM, the data is read from a socket.

### 4.3.2 How the Mpygmalion software should be used

The software for the MPEG-card is interrupt-driven. In the original program the software first initializes the MPEG-card by resetting the chips on the card and installs the interrupt-routines. The program should be started with at least one parameter. This parameter, when running the program to view a MPEG-file on harddisk, is the name of this file and the path where to find it. After that, the MPEG-card is started, decoding the given MPEG-file, and a loop continuously checks the actions the user takes and handles these actions by invoking certain procedures that take care of these actions.

In the software for the set-top box an environment should be created which uses the procedures to handle users-actions. Thus parts of the Mpygmalion-card software are integrated in the software that handles the Man-Machine-Interface between the server and the subscriber. The following original routines of the Mpygmalion-card are integrated in the main-program of the set-top box software:

```
mpeg_setcdrom()
iso11172_stream_interpreter()
audio_volume()
video_last_error()
ts3400_move_disp()
iso11172_stream_command()
```
4.4 Properties of the set-top box

In Section 2.5, several options for the implementation of the set-top box were given. Chosen is to have a set-top box of which the actions are controlled by the server, thus placing the intelligence at the server. The main process to provide a user with access to a Video-On-Demand service (and maybe also other services) runs at the VOD-server. The set-top box is seen as the interface part between the subscriber to the service and the VOD-server itself. Therefore, several properties of the set-top box can be laid down.

- The set-top box should handle the Man-Machine-Interface between the Video-On-Demand service running on the server, and the subscriber. This means that all the screens the user gets to view, and that forms the Man-Machine-Interface, are sent by the server towards the set-top box. There are two types of screens that the set-top box should be able to handle:
  - text screens
  - menu screens

The screens are divided into two kinds. The reason for this is the way the set-top box handles the screens sent by the server and the way the set-top box handles the subscribers actions on these screens. For instance, when a text-screen is shown, the only actions the subscriber can take are to go to the next screen (or previous screen) or to exit the VOD-service. When a menu-screen is shown, the subscriber can choose from several items listed.

Responses to the screens should be sent towards the VOD-server. These responses from the subscriber form the interactive part of the system, and make use of the two-way capacity of the underlying CATV network. To limit the information sent towards the VOD-server, some constraints are made in possible responses by the subscribers. For instance, when a menu is shown where several actions of the subscriber are possible, only permitted responses corresponding with the choices offered to the subscriber are sent towards the VOD-server.

- An important function the set-top box should perform is the decoding of the MPEG-video data. The decoding of the MPEG-data demands the highest priority of the set-top box because this should be done real-time, having a sustained dataflow, thus without any fluctuations. The original software of the MPEG-card has a simple user-interface and is only designed to play MPEG-video. This software is integrated in the software for the set-top box, because the decoding of the MPEG-video is just one function of the set-top box. When a subscriber selects a movie, the set-top box waits for the server to send the MPEG-video and then starts decoding the MPEG-video with the interrupt-driven MPEG-software modules. When the subscriber presses any key, depending on the key pressed the set-top box reacts on the key and sends this action to the server, or identifies the pressed key as being an unexpected keypress and takes no action.

- The User-Interface of the set-top box is an aspect of the system that to a large extent decides the success of the VOD-system. As this is the beginning of a larger project, and time is limited, the User-Interface is kept relatively simple. Communication between the user and the service is done through text screens on the PC-monitor of the set-top box, containing menus or information. The user can act on these screens by using the keyboard. The keyboard simulates a remote-control by only using a limited number of keys the user may press.
4.5 Working of the set-top box software

In this section the working of the software of the set-top box is explained. In figure 4.1 a block scheme of the software is given.

The block scheme is a top/down-design, which should be read from left-to-right and from top-to-bottom. From left-to-right the time of execution is given, from top-to-bottom more details are given. At the top-level, the main function blocks are given, which are then divided into more detailed blocks.

The software basically consists of four blocks:
Chapter 4 Implementation of the Set-Top box

- initialize_environment()
- setup_connections()
- serve_user()
- deinitialize_environment()

The first block initializes the set-top box environment, and shows a welcome-screen that is stored on harddisk at the set-top box. After the initialization, the connections with the server are made, using TCP/IP protocols. First, two UDP sockets are created to form the bidirectional up/down-channel between the server and the set-top box for sending control-messages back and forth. One UDP-socket is used to sent messages towards the server, the other is used to receive messages from the server. Then the set-top box waits for an acknowledge from the server, and creates a TCP connection to form the downward-channel. The set-top box waits for the server's acknowledge so the server can take its time to initialize the new subscriber. Next comes the main block, serve_user, which takes care of the user's actions on screens and the showing of the MPEG-decoded video. This is done in a loop, which stops at an error-state, or when the user chooses to exit from the set-top box. The last block, deinitialize_environment, restores the old environment and closes the connections between the server and the set-top box. This block is reached when the user wants to exit from the VOD-service.

The main block: serve_user

The main loop basically consists of two blocks:

- wait for server action
- handle server action

The idea behind this is that the server is in control of the User-Interface, and should therefore decide the users screen layout and handle the users reactions to the screens. The set-top box waits for the arrival of a message from the server, indicating that a new screen is ready to be read at the set-top box. When a message arrives, the set-top box reads the message and decodes this message. The message can be either a message indicating the arrival of a screen, or indicating the arrival of a film.

When a screen is indicated to arrive, the set-top box reads the screen from the downward channel and makes the screen visible on the PC-monitor of the set-top box. The screen to be shown at the set-top box is sent as an ASCII-file, containing the text to be shown and the location of each line at the screen (see Section 4.5.1 for the structure of the screen files). After the screen is shown at the monitor of the set-top box, a loop is initiated to handle the users actions, reading the users responses from the keyboard. If the user pressed the exit-key (the escape-key), the set-top box is closed down. Otherwise, if a legal key is pressed, the key is sent towards the server that acts on the key. In the message from the server to the set-top box indicating the arrival of a screen, a keymask is added to indicate the set-top box of the possible keys the user can press for this screen (see Section 4.5.3).

When a film is due to arrive, the adapted Mpygmalion software is started. First the MPEG-card is initialized, then the decoding of the MPEG-data is started, reading the MPEG-data from the downward-channel. For this, the MPEG-data is supposed to be available at the downward-channel, thus being sent as a sustained dataflow towards the set-top box. The decoding of the MPEG-data is done using interrupts, therefore the handling of the users actions, the next block, is a loop lasting until the movie has ended, or until a user action exits the showing of the movie. The loop is interrupted by interrupts generated by the Mpygmalion-card and the Mpygmalion interrupt-routines are then handled. When a Mpygmalion-card command is read from the keyboard, actions
are taken at the set-top box concerning the playing of the movie by the Mpygmalion-card. These Mpygmalion-card commands can be: arrow-keys to move around the display window, or the gray +/- keys for adjusting the framerate of the decoding. When the exit key is pressed, the playing of the MPEG-data is stopped, the server is informed with a message sent from the set-top box and the set-top box is closed down. When a user presses a movie playback control key like pause or stop, the set-top box reacts to this by stopping or pausing the MPEG-card and sends a message with the pressed key towards the server. At the moment only the pause and stop playback functions are implemented, no other trick-states are available yet.

4.5.1 Structure of screen-files sent from the server to the set-top box

The screens sent from the server to the set-top box are ASCII-files. A screen file can be used for text-screens or for menu-screens. The structures of these files are identical. The files contain the lines to be shown at the screen of the set-top box, and the position to show them. This is done by giving the $x$ (linenumber) and $y$ (colomnumber) position and the text for each line to be shown on the screen. An example of the openings screen is given below.

1 9 " Welcome at: "
3 5 " The Video-On-Demand Service "
5 5 " Presented by: "
8 5 " The Eindhoven University of Technology "
10 5 " September 1995 "
15 5 " Press 'NEXT' "

4.5.2 The message structure

There are two kinds of message-sets, those sent from the server towards the set-top box and those sent from the set-top box towards the server.

**Messages from server to set-top box**

The structure of a message is given below.

<table>
<thead>
<tr>
<th>command</th>
<th>space</th>
<th>parameter0</th>
<th>space</th>
<th>parameter1</th>
</tr>
</thead>
</table>

The length of command, parameter0 and parameter1 is 10 characters (ASCII), separated by a space. The commands that can be sent are given in table 4.1.
Chapter 4 Implementation of the Set-Top box

Table 4.1: Messages sent from server to set-top box

<table>
<thead>
<tr>
<th>command</th>
<th>parameter0</th>
<th>parameter1</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_SCREEN</td>
<td>size of file</td>
<td>keymask</td>
<td>Parameter0 contains the size of the file being sent from the server needed for the reception of the file with the TCP protocol, and a keymask indicating the possible sets of keys the user may use</td>
</tr>
<tr>
<td>CMD_OK</td>
<td>no param.</td>
<td>no param.</td>
<td>no parameters are added</td>
</tr>
<tr>
<td>CMD_FILM</td>
<td>size of file</td>
<td>keymask</td>
<td>Parameter0 contains the size of the file being sent from the server needed for the reception of the file with the TCP protocol, and a keymask indicating the possible sets of keys the user may use</td>
</tr>
</tbody>
</table>

Messages from set-top box to server

The structure of this kind of message is given below.

```
client ID space command space parameter0 space parameter1
```

The length of client ID, command, parameter0 and parameter1 is 10 characters (ASCII), separated by a space. The commands that can be sent are given in table 4.2.

Table 4.2: Messages sent from set-top box to server

<table>
<thead>
<tr>
<th>command</th>
<th>parameter0</th>
<th>parameter1</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>no param.</td>
<td>no param.</td>
<td>no parameters are added</td>
</tr>
<tr>
<td>KEYS</td>
<td>keymask</td>
<td>key</td>
<td>Parameter0 contains the keymask indicating the possible sets of keys the user pressed, while parameter1 contains the pressed key</td>
</tr>
<tr>
<td>EXIT</td>
<td>no param.</td>
<td>no param.</td>
<td>no parameters are added</td>
</tr>
</tbody>
</table>

4.5.3 Possible keys the user may use

The server indicates the possible keys the user may press for each screen sent towards the set-top box (apart from the keys the user can press to control the Mpygmalion-card, like adjusting the framerate and moving the display around). This is done to limit the communications between the set-top box and the server when unused keys are pressed by the user. For this a keymask is used, which is defined so that sets of keys (like all numerical keys) or individual keys can be checked at the set-top box. The keymask uses flags for each key or set of keys. Possible keyflags are:

- KEY_NUM, indicating all numerical keys
- KEY_NEXT, indicating the NEXT key
4.5. WORKING OF THE SET-TOP BOX SOFTWARE

- KEY_PREV, indicating the PREV key
- KEY_STOP, indicating the STOP key
- KEY_PAUSE, indicating the PAUSE key
- KEY_START, indicating the START key
- KEY_EXIT, indicating the EXIT key

There are three kinds of possible sets, for text-screens, menu-screens and during playback of a movie. The keymasks for these sets are calculated by or-ing the flags of each individual key or key-set:

- text: KEY_NEXT or KEY_PREV or KEY_EXIT
- menu: KEY_NUM or KEY_PREV or KEY_EXIT
- movie: KEY_STOP or KEY_PAUSE or KEY_START or KEY_EXIT
Chapter 5

Implementation of the Video-On-Demand Server

In this chapter the implementation of the server for the Video-On-Demand service is described. First a description of the hardware and the software will be given, after which the structure and the implementation will be described.

5.1 Hardware and software setup

The main goal of any server is to serve clients, providing the clients with one or several (different) services. The hardware for the initial state of this project should be capable of offering a Video-On-Demand service to a few clients, starting with just one client, with software written so that additional clients only have to be connected to the network. The server offering a Video-On-Demand service has to perform several tasks, for instance network-processing and client-handling. When multiple clients have to be served, some kind of multitasking has to be implemented in the software environment. This means programming the server can best be done in a multitasking operating system. There are several options for programming in a multitasking operating system. For instance: OS/2, Windows 95 and Windows NT for PCs, and UNIX. When an operating system had to be chosen for this project, several aspects were considered. For instance: availability of software for this operating system, the knowledge available for help with programming under this operating system, the quality of the operating system and the future prospect of the operating system. Chosen to use as an operating system is Windows NT, because of availability of proper software, the Windows like environment of which expertise is available; the stability of the system; and the fact that Windows NT is specially designed for network purposes. Furthermore, because the set-top box is also a PC, software can easily be exchanged between the computers. Windows NT is also chosen instead of OS/2 and Windows 95 because for these operating systems, software seems to be harder to obtain, less expertise is available and future prospects are unsure. Windows NT is chosen above UNIX because of available expertise and the familiarity with Windows programming.

The compiler chosen to program in is Microsoft Visual C++ 2.0. This is a compiler especially designed for programming under Windows NT, allowing programming in a 32-bit environment with multitasking properties. The server-software also has to implement a socket interface to communicate with the set-top boxes of the subscribers. For programming the server and for the implementation of the socket interface the Win32 Software Developers Kit (SDK) is used, which implements the standard BSD socket interface and also an extended windows-oriented socket interface, based on the Windows-Message-Handling-System (see Section 5.2). The SDK sockets interface makes use of the TCP/IP standard provided by Windows NT.
The server is first implemented on a 486DX66 computer with 16MByte RAM, and a 1GByte harddisk to store MPEG-data. The reason a 486 is used is because of the availability of this setup, and the fact that first a VOD-system for one subscriber is implemented. Also added to the hardware setup are a CD-ROM drive for installing software and copying MPEG-data from CD-ROM to harddisk, and an Ethernet-card (3Com-509 EtherLink III) for the connection with the set-top box(es).

5.2 The TCP/IP implementation

The Software Developing Kit for Windows NT comprises a *Windows Sockets Application Programming Interface (API)*. This interface offers a network programming interface for Windows. The interface is based on the Berkeley Software Distribution (BSD) socket interface. The Windows sockets API contains all the standard BSD functions, and also a set of extensions that provide extra features for Windows programmers.

The extra features offered by the socket interface is the possibility to avoid *blocking*. Blocking takes place in the standard BSD system calls, when for instance data has to be read from a socket. When data has to be read, the system call used to read the data blocks waits until the data is there to read. With the extra features this is avoided by making use of the Windows-Message-Handling-System (see [Petz92] for a more detailed explanation of windows programming using messages). Now *non-blocking* system calls are used. For a more detailed explanation of the non-blocking calls, see [Andr94] and the online help functions available for Microsoft Visual C++.

The non-blocking system calls are used for implementing the upward-channel for sending command-messages from the set-top box towards the server. The upward-channel is implemented by a socket that receives all the UDP-messages sent from any set-top box towards the server. The reception of UDP-messages takes place with the Windows-Message-Handling-System. After creating a socket for the upward-channel the properties of the socket are changed so that when the socket receives an UDP-message, a Windows-message *WSA_READ* is generated. The Windows-Message-Handling-Loop receives the WSA_READ message and than starts handling the reception of the message.

For each subscriber two other sockets are created. One socket is used as an UDP-socket for sending command-messages from the server towards the set-top box. The other socket is used as a high-speed downward-channel and is used as a TCP-socket.

5.3 Multitasking

One of the features of Windows NT is *preemptive-multitasking*. With this feature several software applications can run at the same time, and a malfunction in one application cannot lead to unexpected behaviour in another application. Preemptive-multitasking is implemented with a method called multithreading. This is a technique that divides processes in threads, and allocates each thread a share of CPU time.

In Windows NT a thread is the smallest entity that can receive CPU time. A process in Windows NT is a task that a developer uses for a program. It can consist of multiple threads. Each thread can perform a function in the process.

A process always consists of a *main-thread* which starts the process. From this main-thread, other threads can be spawn, each performing a separate function. One thread could for instance be used for real-time I/O, while other threads could be doing calculations, or background printing, et cetera.
In a single processor computer, multithreading does not improve the execution speed of an application, it can even slow it down if too much threads have to be kept track of. When a multiprocessor computer (starting from two processors) is used, an increase of execution speed can be obtained, depending on the way the threads are managed.

When multithreading is used in an application, it might very well be possible several threads have to wait for each other or work together, sharing data that only one thread at any time should be able to change. This calls for synchronization methods between threads. There are several methods for synchronisation purposes:

- **Mutex objects**, which can protect shared resources (data) by ensuring that only one thread has access to those resources.
- **Semaphore objects**, which can limit the number of threads that can be executed simultaneously.
- **Event objects**, which can prevent a thread from starting until the execution of another thread is complete.

First the concept of objects should be explained shortly for better understanding of programming under Windows. Programming in Windows NT is done using objects (these are Win32 SDK objects, not C++ objects used in Object-Oriented-Programming) and handles by which objects can be addressed. An object is an internal structure that represents a system resource, such as a file, a thread, or a mutex. Handles are pointers to objects. For more information about objects and handles, see [Petz92,Andr94].

In the server application mutex objects are used for synchronization purposes. The mutex objects are waitable objects. A waitable object is an object that can be in a signalled or an unsignaled state. If an object is under use, for instance by a thread, the object is in signalled state. Other threads can use a Win32 function called: 'WaitForSingleObject' which waits until the object returns to unsignalled state. The object returns to unsignalled state by releasing the object (the mutex in this case). When the mutex is released, the waiting thread takes control of the mutex and continues with what it was doing.

### 5.4 The structure of the User-Interface

The User-Interface of the set-top box is controlled by the server and is built up by screens containing text and menus. The subscriber using the set-top box reacts to these screens by using the keyboard to make choices, and these users-reactions are transported to the server by using messages. The server handles these users-actions. For this project an example User-Interface is used given in figure 5.1. At the start of a session between the subscriber and the server an openings screen is shown at the set-top box. When a user presses the 'NEXT'-key, a menu screen is shown at the set-top box, offering the user a choice to go to **Personal Management**, **Movies** or **News**. The first and last choice are not further implemented, and lead to a text screen indicating this. When **Movies** is chosen, a film menu is shown, offering two film choices.

The User-Interface for subscribers is implemented by using a linked-list. The linked-list contains all the possible user-actions that can be taken, all possible screens that can be shown, all movies that can be offered to the subscriber, and the locations of the screens and movies on the local harddisk of the server. To create the linked-list a file containing the structure located at the server is used. In this ASCII-file the structure of the User-Interface is given through command-words. To initialize the linked-list the file is decoded. This decoding is done by reading

1 The file can be found at the path C:/THESIS/STRUCT/struct.ods
Welcome at:
The Video-On-Demand Service
Presented by:
The Eindhoven University of Technology
September 1995
Press 'NEXT'

Welcome at the On-Demand Service
These are the services we offer:
1. Personal Management
2. Movies
3. News
Please make your choice

You can choose from the movies listed below.
MOVIES:
1. Nike commercial
2. NOB commercial
Choose one of the numbers.

This service is not yet available
It will be soon.
Please try again later
Press 'NEXT'

This service is not yet available
It will be soon.
Please try again later
Press 'NEXT'
the ASCII-file commands and creating and appending the linked-list blocks. See Appendix A for the contents of the applied structure-file. The linked-list, implementing the User-Interface (of figure 5.1) is given in figure 5.2.

![Linked-list of example User-Interface.](image)

The linked-list can contain several kinds of blocks. Each block contains the following standard values:

- **num**: an unique block number (in the list)
- **type**: the type of block
- **file**: name of the file of the screen to be sent towards the set-top box, or the name of the film to be sent
- **filelength**: length of the file to be sent towards the set-top box. Used for reading the proper length from the socket at the set-top box.
- **keymask**: indication of the keys the user may press at the set-top box
- **next**: pointer pointing at next block in the list

In addition, a menu block also contains an **itemlist-pointer**. This is a pointer to an item list. The item list is another linked-list, which contains blocks with the possible choices a user may make, and the new block to jump to when that choice has been made. Each item therefore contains a key-value and a number of the block to jump to when this key has been pressed.

As mentioned before, there are several types of blocks. These are: **start**, **menu**, **text**, **film**, **menu** and **film**. The **start**-block is used as an indication of the first screen to be sent towards the set-top box. There should always be a start-block. Text-blocks and film-blocks are blocks indicating text-screens, and films respectively. The **menu**-blocks indicate menu-screens. There are two kinds of menu-screens: **menu** and **film-menu**-screens. At the moment these two kinds of menus are not treated differently in the software, but the difference is there for possible future changes in the User-Interface. For instance when the User-Interface is not implemented by ASCII-files but with MPEG-encoded still-pictures. Then it might be possible that a standard film menu screen is available, with a flexible list of movies to be offered stored in a different format, i.e. text files. For this, separate menu types are implemented.

The files used as screens are ASCII-files located at the server. The files contain the text to be shown at the screen of the set-top box, and the position of the text. See also Section 4.5.1.
5.5 The working of the server software

The software of the server is written under Windows NT, making use of the multitasking qualities of Windows NT. The programming is done using threads, creating a multithreading application.

Programming under Windows means using the Windows Message-Handling-System. To get an idea of programming for Windows, studying [Petz92, Andr94] is advised. The Message-Handling-System on which Windows is based, has events sending messages to a main-loop running when the Windows-application is started. For instance, the server-application creates a full-screen window that has standard window-properties, like the possibility to resize the windows, turn it into an icon, et cetera. There is also a menu at the top of the window, which offers the chance to exit the server, by choosing Exit, or for instance an About choice, to get a small window with some information. Whenever a choice like this is made, a message is sent to the Message-Handling-System indicating the action, after which a subroutine in the server-software takes care of this message. It is also possible to create software-events where an event-occurrence causes a message, or to send messages from within, for instance, a thread. These messages are all handled by the Message-Handling-System. Messages can be defined by the programmer, or can be standard messages.

The server-application is built up in the following way. There is a main-thread that continuously listens to incoming messages from set-top boxes, and for each set-top box connected to the server there is one or several threads serving the subscriber. These threads used for serving the subscriber, handles the User-Interface of the subscriber, sending screens to the set-top box, and transporting MPEG-movies towards the set-top box. The main-thread has its main-function in receiving the messages from the set-top boxes and distributing the received keys pressed by the subscriber towards the threads handling the subscriber. For this last part, mutex programming is used.

When a main-thread initializes the thread for handling the subscriber it takes control of the subscribers specific mutex (each subscriber has its own uniquely named mutex when connected to the server). The thread handling the subscriber waits for the release of the mutex by the main-thread, indicating the reception of a key pressed by the subscriber. The thread handling the subscriber takes control of the mutex and reads some global data where the specific key is stored. When the action of the subscriber is handled, it releases the mutex again. The main-thread again takes control of the subscriber-specific mutex.

5.5.1 The main-thread

The main-thread of the server is based on the Windows-Message-Handling-System. This system receives software-messages and handles these. The structure of the main-thread will be explained with a top/down-scheme, given in figure 5.3.

When the application is started, the first message the Message-Handler receives is a WM_CREATE message. When this message is handled, the server initializes the server. This is shown in figure 5.3 by the top-most left block. When the server-application is started, an UDP-socket is created. This socket is used to listen to incoming messages from any set-top box. For this the Message-Handling-System is used, making the socket non-blocking with the WSAAsyncSelect-socket-call. Whenever a message is received by this socket, a WSA_READ-message is created. The Message-Handling-System, shown in the second top-most block, takes control of the server-application, being a loop until the server-application is ended. In this loop there are several messages that can be received which will briefly be explained here. First, and the most important function of the main-thread, is the reception of a WSA_READ-message, indicating the arrival of an UDP-message from a set-top box as shown by the block 'handle WSA_READ'.
Figure 5.3: Main-thread of server application

1. Initialize server
2. Create window
3. Make UDP upward socket
4. Wait for Windows msg
5. Case WSA_READ
   - Read and decode message
6. Case WM_DESTROY
   - Switch (message)
7. Get mutex of client
8. Case IDM_GET_MUTEX
9. Case CMD_KEYS
   - Send message to client_process
10. Case CMD_EXIT
    - Remove client
11. Case CMD_INIT
    - Check for existence client
    - Make UDP downward socket
    - Create child window
    - Add client to clientlist
    - Create mutex for client
    - Initialize user-interface structure
    - Start client_process thread
5.5. THE WORKING OF THE SERVER SOFTWARE

When an UDP-message from a set-top box is received, the message is read and decoded, to
determine the nature of the message. The messages that can be sent between the set-top box
and the server are explained in Section 4.5.2. There are three kinds of UDP-messages that can be
received: CMD_INIT, CMD_KEYS and CMD_EXIT. When a CMD_INIT is received, a set-top
box indicates that it wants to be served by the server and a session should be started. To handle
that, first it is checked if the set-top box isn't already connected to the server. Every set-top box
connected to the server is added in a clientlist at the server. This clientlist is a global linked-list
with blocks containing data about every set-top box, such as the state it is in, the addresses of the
set-top box so data can be transported towards the set-top box through sockets, et cetera. Every
client (set-top box) in the list has its own unique number, which is also sent with all messages
from the set-top box towards the server.

If it is determined that the set-top box is not in the clientlist, an UDP-socket for sending data
towards the set-top box is created. This socket does not use the special non-blocking features, but
is a socket according to the standard BSD-socket interface. Next a window is created to indicate
a new client at the server-side. At the server the window is shown inside the main-window as a
small client-window. All the data are then added in the clientlist.

The next block in the scheme, 'Initialize User Interface', creates the structure-linked-list for
the subscriber, containing all the user-actions, screens to be shown at the set-top box, et cetera.
There are several options for using the structure-list.

1. one global structure list for all subscribers. Here there is one linked-list created, used to serve
all subscribers by pointing to this list and using the blocks in this list.

2. one linked-list for every subscriber. Here each subscriber has a linked-list with the structure
in it.

The advantage of the first method is the fact that less memory is used to store the User-
Interface. The advantage of the second method is the flexibility when the User-Interface should be
changed during sessions of subscribers, when subscribers are already using the linked-list. This is
only important when several subscribers can be connected, thus at the moment it does not play a
significant role, but in the future when more subscribers are handled, it can be important to have
a flexible structure. If the structure should be changed, the subscribers can continue to use their
own, complete User-Interface. Chosen is to use the second method and give each subscriber an
own linked-list because of the flexibility, and to live with the fact that slightly more memory is used.

After the linked-list is created, as shown by the block 'create mutex for client', a mutex is
created for the specific set-top box, with its own (unique) name by which it can be addressed. The
mutex is used as a signaling-object to indicate the thread handling the subscribers User-Interface
that a key has been pressed. When the mutex has been created and the main-thread has taken
control of the mutex, the thread for handling the set-top box is created, as shown in block 'start
client-process thread'. This thread now handles the User-Interface of the subscriber with the
server. The main-thread returns to the Message-Handling-System.

5.5.2 The thread handling the set-top box

The thread for handling the set-top box, the client process, has to serve the subscriber using the
set-top box, and does this according to the User-Interface laid down in the structure linked-list.
The working of this part of the software is again explained by a top/down block scheme, as shown
in figure 5.4.

The client-process consists of three blocks (top-most level). These are 'create TCP downsocket',
'handle structure of user-interface' and 'handle user management & exit'. The first block creates
the TCP downsocket used to send screens and MPEG-data towards the set-top box. The second
handle user response

handle user action

wait for user action

get item from file

is item a new handshake block?

yes

no

handle new handshake block

put item to set-top box

wait for user action

create TCP down socket

handle block

TCP to set-top box

send file with

wait for user action

handle new handshake block

put item to set-top box

wait for user action

handle block

TCP to set-top box

send file with

wait for user action

handle new handshake block

put item to set-top box

wait for user action

handle block

TCP to set-top box

send file with

wait for user action

handle new handshake block

put item to set-top box

wait for user action

handle block
block, and the most comprehensive, handles the structure of the User-Interface for serving the subscriber. The last block is not implemented yet, but could handle management function like account-management, payments, statistical functions, et cetera.

Handling the structure of the User-Interface

The handling of the User-Interface is the main-part of the client-process. This is done using a loop that exits when the set-top box is closed down because the subscriber wants to exit from the service. It consists of five blocks, the second level in figure 5.4.

In the loop a pointer is used which points at a structure block in the linked-list of the structure. The first time the loop is executed, the pointer points to the first structure block in the linked-list, the start-structure block. The first block as shown in figure 5.4 handles the structure block the pointer points to. If the block is a text screen or a menu screen, a screen file with the text or menu is sent towards the set-top box to be shown at the monitor at the set-top box. This is done by sending an UDP-message (see Section 4.5.2 for the contents of the message) towards the set-top box to indicate the arrival of a TCP-file. Then the screen-file is sent over the downward TCP-socket.

After the structure block is handled, the client-process waits for a key pressed by the subscriber. This is done by waiting for the release of the mutex for this set-top box by the main-thread. As long as no key has been pressed and no message containing the pressed key has arrived, this thread waits. When the mutex is released, it continues.

When some keypress has arrived at the client-process thread, two things can happen. If the exit-key has been pressed, the handling of the structure is stopped. If another response is given by the subscriber, depending on the screen the subscriber responded to, two things can happen:

• if a text screen was responded to, the pointer to the previous structure block of the linked-list is changed, and the linked-list is searched to find the structure block with the number the previous structure block points to.

• if a menu screen was responded to, several things happen. First the itemlist of the menu structure block is searched for the key the subscriber pressed. If the key is found, first it is checked if the item pointed to is a film. If this is not the case, the pointer of the linked-list is changed to the next structure block. If the item is a film, the film is handled (see the following subsection).

After the key pressed by the subscriber has been handled, the mutex is released again, and the main-thread is signalled to take control of the mutex, by sending an IDM.GET_MUTEX-message to the Message-Handling-System of the main-thread, where this message is handled.

The handling of a film

The handling of the film is done using another thread. The reason another thread is used is because the client-process has to wait for the release of the mutex indicating the arrival of a key, and the transport of the MPEG-data of the film should be sustained.

First a thread is started which sends the MPEG-data towards the set-top box. Then the client-process starts a loop that checks for subscribers responses. This loop is similar to the loop shown in the second level at figure 5.4. It waits for the release of the mutex. At the moment the software only allows the playback of the movie chosen at the set-top box. No trickstates have been implemented.
Chapter 6

Conclusions & Recommendations

This chapter describes the conclusions and the next steps that could be taken for continuing the Video-On-Demand project, and enhancing the hardware and the software of the demonstration network.

6.1 Conclusions

Video-On-Demand is an area on which a lot of research work is done all over the world. There are some general aspects of Video-On-Demand that are very important to realize for everybody active in this area. It is very important to realize what Video-On-Demand demands of the network and the hardware:

- Video-On-Demand uses movies stored in digital format, encoded according to the MPEG-standard. If MPEG-1 is used, an average movie has a size of 1GBytes. When MPEG-2 is used, the size of an average movie increases with a factor two to four, thus a size ranging from 2-4GBytes.

- At the moment the project uses a datarate of approximately 1.5Mb/s according to the MPEG-1 definition. The MPEG-1 definition requires a sustained datarate. The new standard for multimedia CD’s will be the MPEG-2 definition, which has a flexible datarate. This changes the requirements on the underlying network.

Following are some conclusions about the implementation of the Video-On-Demand system:

- The maximum datarate obtained using the TCP/IP stack was 370kBytes/s. This speed was reached by writing data towards the TCP-socket at the server-side continuously, and reading the data at the client-side continuously. The limiting part of the software probably lays in the 3Com TCP/IP stack.

- The source-code has been adapted so that it can be used in combination with the 3Com TCP/IP stack and that it is possible to read the MPEG-1 data from a socket-connection, instead of reading from a standard information-source like a CD-ROM.

- MPEG-1 encoded video can be sent from a server towards a client (the set-top box) over a TCP-socket connection, where the client decodes the information to regain the original analogue video.

- At the moment it is possible to connect one set-top box to the server. The software for the server is written so that it is possible to connect several set-top boxes to the server. The
6.2 RECOMMENDATIONS

The number of set-top boxes that can be connected to the server is limited by the Ethernet-cards and the TCP/IP-standard used for the connection. The Ethernet-cards used have a maximum datarate of 10Mb/s, thus theoretically seven set-top boxes could be connected through one Ethernet-card at the server. There is however overhead introduced by the TCP/IP stack, which will limit the number of set-top boxes further. If the maximum datarate obtained is looked at, it is only possible to connect two set-top boxes. But when more set-top boxes are connected a parallel-TCP-channel is obtained, which could mean a higher number of set-top boxes is possible when the datarate the server can send is higher than the confined datarate of the 3Com stack.

- The MPEG card used is not 100 percent stable. Sometimes it hangs itself when decoding the MPEG-1 data. This is a vault in the original MPEG-card source-code, maybe introduced when the source-code has been rewritten for large memory model. An other more serious problem is the fact that after a movie has been played, it is not possible to reset the MPEG card and decode another MPEG movie. This fault is also present in the original software when the repeat option is chosen when playing a MPEG movie.

6.2 Recommendations

Any Video-On-Demand project is a large project, the project of the demonstration network also. There are thus many subjects in need of more research. The recommendations can be divided in three parts, recommendations for the set-top box, the server and general recommendations.

6.2.1 Recommendations for enhancing the set-top box

The present implementation of the set-top box is kept relatively simple. The User-Interface of the set-top box is a topic of which much research is still going on all over the world. At the moment the User-Interface uses the monitor of the set-top box PC. The graphical layout is implemented with plain text structures, showing text-screens and text-menu’s. Choices at menus is done by using the keyboard. The User-Interface could be further enhanced by changing the screens used for the User-Interface from text-screens to MPEG-encoded still-pictures, where the set-top box decodes and displays them as still-pictures. This way, the monitor of the set-top box PC can be discarded. A further enhancement can take place by using a remote control unit instead of the keyboard. Furthermore the MPEG card has to be looked upon, because of the occasional hangups, and more serious the fact that it is (at the moment) not possible to play several MPEG movies one after another.

6.2.2 Recommendations for enhancing the server

The server is also a subject where further enhancements can be made.

- At the moment the server can let a subscriber go through a menu-based User-Interface to make a choice of movies, which are then played. No VCR-functions are implemented, except pause. To have a fully interactive Video-On-Demand system, VCR-functions should be provided. Implementing certain VCR-functions like fast-forward/reverse implies manipulating the MPEG-datastream, by either sending the datastream at a higher datarate towards the set-top box, or by only sending those I-pictures of the MPEG-datastream that are separated by a certain amount of time, for instance 10 seconds. The first method is not possible considering the underlying network and the constraints it would lay on the network. Therefore, only the second solution is possible. The second method, manipulating the MPEG-datastream can both be done when the MPEG-data is read from harddisk (or another medium) or when buffers are used to transport the MPEG-data towards the set-top box. When buffers are used the system becomes more complex. Either the data that is sent to the buffer at the
server is manipulated, or the data transported from the buffer to the set-top box is manipulated. When the MPEG-2 definition is used trickstates are even more complex because of the flexible datarate.

- At the moment one set-top box has been connected to the server. The server is built to serve more set-top boxes. The bottleneck of the amount of set-top boxes the server can serve lays in the Ethernet-cards. Further research is necessary to get an idea of the number of set-top boxes that can be connected to the server. A possibility to increase the number of set-top boxes connected to the server is by adding more Ethernet-cards to the computer. When more Ethernet-cards are used, the software needs to be extended to allow for routing to address the set-top boxes through multiple Ethernet-cards.

6.2.3 General recommendations

- An other possibility to increase the number of set-top boxes that can be served by the server is to use dedicated hardware. The dedicated hardware should be optimised for sending MPEG-encoded data towards a set-top box. A structure for this dedicated hardware is given in figure 6.1.

![Figure 6.1: Example of dedicated hardware structure](image)

Here the internal data bus is used in an optimal way, offering a datarate of 32MBytes/s when a PCI-bus is used in combination with a Pentium computer. The data bus can be used to transport data to four dedicated hardware-cards. These cards can contain for instance RAM-buffers that are filled with data that should be transported to individual set-top boxes. The dedicated hardware-cards could have their own clock-systems for synchronisation and transport of the data towards the set-top boxes, independent of the computer. It might be necessary to use two computers. One dedicated for routing purposes, making optimised use of the dedicated hardware, and one computer for receiving and sending messages from and towards the set-top boxes. When dedicated hardware is chosen, it is advisable to step away from the TCP/IP protocol, and use a protocol that takes into account the specific properties for transporting digitalised MPEG-encoded video. The key issue here is to support sustained transport of the MPEG-data.

- This project uses MPEG-1 encoded video. It is recommended that for future use the MPEG-2 standard is concentrated on, because in september/october 1995 MPEG-2 has been chosen by international committees as the new multimedia video-standard for encoding digital video.
This implies higher datarates of the underlying networks, and a way to take into account the flexible datarate for decoding the MPEG-2 data.
Appendix A

The structure of the file used to build the structure-linked-list

In this appendix the file "C:/THESIS/STRUCT/STRUCT.ODS" is given. This file is used to build the linked-list containing the structure of the User-Interface.

start
file "C:\THESIS\TEXT\opening.txt"
filelength 177
keys NEXT
keys EXIT
goto 100000
end

menu 100000
file "C:\THESIS\MENU\menu1.txt"
filelength 176
keys NUM
keys EXIT
menuitem TEXT 200000
  keys 1
end
menuitem MENSFCILM 300000
  keys 2
end
menuitem TEXT 200001
  keys 3
end
end

text 200000
file "C:\THESIS\TEXT\notavail.txt"
filelength 121
keys PREV
keys EXIT
end

text 200001
file "C:\THESIS\TEXT\notavail.txt"
The structure of the file used to build the structure-linked-list

filelength 121
keys PREV
keys EXIT
end

film menu 300000
  file "C:\THESIS\MENU\menu2.txt"
  filelength 164
  keys NUM
  keys PREV
  keys EXIT
  menuitem FILM 400000
    keys 1
  end
  menuitem FILM 400001
    keys 2
  end
end

film 400000
  file "C:\THESIS\FILMS\nike.mpg"
  filelength 5210000
  keys STOP
  keys EXIT
  keys PAUZE
end

film 400001
  file "C:\THESIS\FILMS\film"
  filelength 20836984
  keys STOP
  keys EXIT
  keys PAUZE
end
Appendix B

The address of ELKA Automatisering

The address of the supplier of the Mpygmalion MPEG-1 card is:

Inside Technology,
Fortranweg 7
3821 BK Amersfoort
Tel. 033-562814
Fax. 033-558310

The supplier of the card is Inside Technology. The contact person there is Leon Kohlen, who has his own company called ELKA Automatisering, which develops the Mpygmalion card distributed by Inside Technology. For specific information about the MPEG card, and to get the address of the designers, Leon Kohlen should be contacted. He is the one who knows about the project using the MPEG cards.
Bibliography


