Master Thesis

Survey on video-on-demand broadcasting protocols

Author(s):
Zubimendi, Leire

Publication Date:
2008

Permanent Link:
https://doi.org/10.3929/ethz-a-005565529

Rights / License:
In Copyright - Non-Commercial Use Permitted
Department of Computer Science
Institute for Pervasive Computing
Information and Communication Systems Research Group

Leire Zubimendi

Survey on Video – On – Demand Broadcasting Protocols

Master Thesis
October 2, 2007 – April 2, 2008

Supervising Assistant: Voulgaris Spyridon
Supervising Professor: Prof. Gustavo Alonso
This survey is an analysis about different Video-On-Demand protocols. They have been classified depending on the mode of service that they have. After analysing them, the most popular two developed systems have been explained: Youtube and CNN Pipeline. Video-On-Demand is a service that enables users to select a video (or a movie) from a large collection while sitting at home and permits to have almost as a good a control over the viewing of the video as when using a conventional VCR.
DEDICATION

To my parents
I think that this thesis deserves also a little place for gratitudes. This work started six months ago; it has been a long way that after all this period of hard work at the end has its result. I came last year to Zürich in order to do the Erasmus internship and at the end of my stay I found this project thanks to professor Gustavo Alonso. I would like to express my gratitude to him for giving me this opportunity. I’d like also to say thank you to my supervisor Voulgaris Spyridon. For both of us this thesis has been stressing, especially for him because of the rest of the work that he has to do; but he always helped me guiding me throughout it.

All the students that are studying computer science know that at the end of our degree we have to show our knowledge in the field and make a thesis that will be our first big job. I want to say thank you to my parents for giving me the opportunity to study here and also for their unconditional help and support. Without them this project would be impossible.

I cannot forget Sha and Dani my best friends here. They always encouraged me specially in days in which I was thinking that I could not manage this thesis. Thank you for believing in me. Without their support I would have been lost during my stay.

To finish with this section I cannot forget the person that all this time had to see me laughing, crying, in a bad mood, in a good mood, happy, sad, desperate...A thank you it is just nothing. I have not words to say or write in order to express all my gratitude.
You always trusted me and you were there unconditionally. Grazie mille Suraj.
# TABLE OF CONTENTS:

Abstract  
Dedication  
Gratitudes  

## CHAPTER I  
I.1 Motivation ................................................................. 3  
I.2 Introduction ............................................................... 4 - 5  
I.3 Notation ............................................................................. 6  
I.4 Data Centered vs User Centered ................................................. 7 - 9  
I.5 Conventional Broadcasting .................................................. 10 - 12  

**Section I: Data-centered protocols**  

## CHAPTER II  
II.1 Biasing the size of segment .................................................. 13 - 14  
    II.1.1 Pyramid Broadcasting .................................................. 14 - 21  
    II.1.2 Permutation based Broadcasting ..................................... 21 - 28  
    II.1.3 Skyscraper ...................................................................... 29 - 33  
    II.1.4 Fast Broadcasting ......................................................... 33 - 36  
II.2 Comparison between different models ..................................... 36 - 37  

## CHAPTER III  
III.1 Biasing the bandwidth of segment ......................................... 38 - 39  
    III.1.1 Harmonic Broadcasting ............................................... 39 - 44  
    III.1.2 Cautious-Harmonic Broadcasting .................................. 44 - 45  
    III.1.3 Quasi-Harmonic Broadcasting ....................................... 46 - 49  
    III.1.4 Polyharmonic Broadcasting .......................................... 49 - 53  
III.2 Comparison between different models ..................................... 53 - 54
CHAPTER IV .................................................................................................................55
   IV.1 Biasing the periodicity of the segment transmission............................................55
      IV.1.1 Pagoda Broadcasting protocol.................................................................56 - 58
      IV.1.2 New Pagoda Broadcasting protocol......................................................58 - 59
      IV.1.3 Fixed-delay Pagoda Broadcasting Protocol.............................................59 - 61
   IV.2 Comparison between different models.........................................................61 - 63

Section II: Use -centered protocols

CHAPTER V .................................................................................................................64
   V.1 Static multicast..............................................................................................65
      V.1.1 Batching.................................................................................................65 - 66
         V.1.1.1 Batching by timeout.......................................................................66 – 68
   V.2 Dynamic multicast.......................................................................................69
      V.2.1 Patching.................................................................................................69 - 71
         V.2.2 Adaptive Piggybacking......................................................................72 - 77

Section III: Conclusion & Deployed systems

CHAPTER VI .................................................................................................................78
   VI.1 General comparison & Conclusion..................................................................78 - 80

CHAPTER VII: APPENDIX: DEPLOYED SYSTEMS..............................................81
   Youtube..................................................................................................................81 - 82
   CNN Pipeline........................................................................................................82 - 84
   Bibliography..........................................................................................................85 - 87
Section I: Data-centered protocols

CHAPTER I

1.1 MOTIVATION

Video-On-Demand is a concept that is present in our daily life. Cable television, internet etc. are concepts that some years ago were unknown for a general population but nowadays they are concepts that belong to our routine. The simple fact to going to rent a video or to be waiting for the news in tv are things that disappeared with internet and cable television. The utopia of some years ago is now reality.

Who would have told us that we could watch movies downloaded from internet or news live, or that we would have the possibility to choose the movie that we want, to watch it when we want, and furthermore have a full control over it? This is true now and moreover with the best quality. That is one of the reasons why we choose to do this work. That is why we have thought to analyse protocols used to make this possible. We analyse and compare them to see the advantages and disadvantages that each one has. In the last point of our work we present our conclusion about that we have presented in the thesis.

The client does not pay for a service that does not satisfy him/her and with this thesis we want to improve a service that these protocols offer by minimizing their requirements in order to offer a good service as cheap as possible at the end.
Video-On-Demand is a service that enables users to select a video (or a movie) from a large collection while sitting at home and permits them to have as good a control over the viewing of the video as when they use a conventional VCR. It allows users to watch what they want when they want.

Nowadays when we speak about Video-On-Demand we refer to set-top-boxes, computers, mobile phones or any other system that can receive on demand audio-visual content over the network. This technology was launched in 1990 in Hong Kong but it had not much success because of the lack of good development in that times and the fact that was underpaid while video CDs were still cheap. That made Hong Kong Telecom loose so much money and the service had to be ceased in the end.

In 1998, the first fully commercial VOD service was launched in UK. This service also offered TV Broadcast and Internet access through a single set top box using IP delivery over ADSL but it was not until 2006 when Sky Anytime on PC uses a legal peer-to-peer approach to provide a very high capacity multi point downloads of the video content. Instead of downloading all data from a server, data were downloaded from users of the system who had already downloaded the same content. According to the European audiovisual observatory, 142 paying Video-On-Demand services were operational in Europe at the end of that year, increasing year after year until nowadays.

In VOD different techniques such as broadcast (No-VoD), Pay-Per-View (PPV), Quasy Video-on-Demand (Q-VoD), Near Video-on-Demand (N-VoD) and True Video-On-Demand are used to deliver data. The protocols that we present are Near Video-on-Demand protocols but we will explain here the differences among different techniques that exist. The No-VoD is a technique in which the user is passive and has no any control on what he/she watches. On the other hand, in PPV the viewer signs up and pays to watch a specific program. From the other side, Q-VoD from the other side is a technique that groups users according to their interests and then the user can successively control, for a set of time, what to see depending on the group that switches. The two most interesting techniques for us are Near Video-On-Demand and True Video-On-Demand. Near Video-On-Demand is a service that makes the user to wait to watch the video that he/she wishes not having the full control over it whereas
in True Video-On-Demand the client has the 100% of control over the session and does not have to wait for it.

As we know, the main issues of Video-On-Demand protocols are the bandwidth requirement, the storage requirement and the access time for the user to start watching the video that he/she wishes. We cannot forget the storage capacity because from the moment that there is a difference between the consumption rate and the transmission rate of the video, being this one bigger, we need a storage buffer to collect data that the client will receive before she/he can use it. There are different standards to compress videos. In our work we will use MPEG-2, a standard from MPEG. MPEG is the Moving Picture Experts Group which is a working group of ISO/IEC charged with the development of video and audio encoding standards. Using MPEG-2 the video consumption rate will be 3 Mb/s.

In Video-On-Demand several aspects must be taken into account. Another one is the bandwidth. We want to use the minimal bandwidth from the server side because in this way the amount of costs will be clearly reduced. The more requirements we need to broadcast the video and watch it, the more expensive will be the system and the user will pay more. This is something that cannot be accepted. Furthermore, clients want to have the requested movie as fast as possible. The ideal case would be to have it in the same instant that they request for it.

Referring to the access time of the video, it is obvious that we should try to minimize it. To make customers wait to watch their videos minimizes the advantages of Video-On-Demand. Nobody wants to pay for a service, and have data so late, because it would worth more to watch news in TV or rent a movie in the video club.

To explore all these points we analyse the different protocols that are nowadays in Video-On-Demand and then we make a comparison among them. We separate them into four chapters that are categorized depending on other size of segment, the bandwidth of them and the frequency at which segments are transmitted. In the fifth chapter we explore a group of protocols that are user-centered based. At the end of the work we present systems that are already deployed in internet that use these protocols.
I.3 NOTATION

In this section we show the notation that will be used throughout the thesis:

\( M = \# \text{ of movies} \)
\( L = \text{length of the movie} \)
\( L_i = \text{Length of the segment} \)
\( b = \text{Consumption rate of the video} \)
\( a = \text{Access time} \)
\( C_i = \text{Channel i} \)
\( Z = \# \text{ of logical channels} \)
\( TB = \text{Total bandwidth} \)
\( B = \text{How much time we have the consumption rate of the video} \)
\( S = \# \text{ of segments} \)
\( B' = \text{The bandwidth of the subchannel} \)
\( P = \text{Period} \)
\( T = \text{Waiting time} \)
\( T_S = \text{Wtarting time} \)
\( T_{SL} = \text{Time slot} \)
\( St = \text{Storage} \)

These are the main parameters. Any appearance of a new parameter is explained through the thesis.
I.4 DATA CENTERED vs USER CENTERED

There are three important characteristics that make VoD to be good or bad. They are the bandwidth, i.e. its utilization, the client waiting time and the storage capacity required on the client’s node. These concepts are so important in order to decide if VoD is competitive and cost effective. When we speak about VoD we can have different classifications of protocols depending on the criteria and assumptions that we take into account. In this section we will explain one of the most important feature that is taken into account when we classify them. It is the quality of the service mode.

There are three kinds of services: data centered, user centered and the hybrid that is data + user centered service.

What the user wants is to have the video that he/she has chosen as soon as possible and without any discontinuity in its emission. Depending on the mode of service that we are using we will be more focused on the user or in the requested video. The way in which we organize the bandwidth and storage capacity will also change clearly. In the user centered service the user will have a dedicated bandwidth, although not all the time. To get this bandwidth there are two ways. The first one is to provide a sufficient bandwidth equal to an object consumption rate to an object consumption rate multiplied by the number of users and the second way is to provide less bandwidth. In the latest case the users have to compete with a scheduler. In user centered approach the client has to do a request to the server and the server will send the information via a dedicated channel. The main problem of user centered protocols is that as each client will have a dedicated channel the network will be exhausted.

The data centered approach is more focused on objects having each one some allocated bandwidth on a logical channel. It is also known as Broadcasting. With this mode of service we solve the problem arised in user centered mode. Now channels will be dedicated to videos instead of to users. Data centered mode allows users to share a video stream using multicast and also reduces network and server bandwidth. Thus, this service is more scalable than the user centered. Requests of the same video that are made in the same time interval will be batched and served in the same stream. Data centered algorithm further is divided into client initiated (or client pull) and server-initiated (server push) algorithms.
A client initiated algorithm will allocate channels at the request for the same video object, and it will serve a batch of requests. In client initiated service channels are allocated among the users and the service is initiated by clients. On the other hand, the server initiated will continuously multicast video objects via dedicated channels. Clients join the appropriate channels to receive desired video data. It is called also periodic broadcast or service-push service.

A server initiated algorithm can guarantee the maximum service latency independent of the arrival time of the request. There are several schemes that differ on how the video objects are scheduled for transmission. Furthermore, service latencies can be reduced when clients have the ability to prefetch video data at the same time as they are receiving the video data that is being played out. We refer to these algorithms as Server Initiated With Prefetching (SIWP) algorithms. These algorithms provide the best performance to date because they can support a large number of clients while providing low service latencies.

In data centered the user does not explicitly request for a movie. Requests are not sent to the server. Each video is allocated some bandwidth on a logical channel. This bandwidth is reserved for a periodic broadcast of the video. When the user wants to watch a video he/she just tunes into the channel on which the video is broadcasted and waits until it is broadcasted. On the table below we can observe how different protocols are divided depending on the mode of service that they use.

<table>
<thead>
<tr>
<th>Features</th>
<th>DATA-CENTERED</th>
<th>USER-CENTERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels</td>
<td>dedicated to objects</td>
<td>Channels dedicated to users</td>
</tr>
<tr>
<td>Methods</td>
<td>Pyramid Broadcasting</td>
<td>Adaptive</td>
</tr>
<tr>
<td></td>
<td>Permutation based</td>
<td>Piggybacking,Batching,Patching</td>
</tr>
<tr>
<td></td>
<td>Skyscraper</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Harmonic Broadcasting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fast Broadcasting</td>
<td></td>
</tr>
</tbody>
</table>

*Table 1: Summary of Data-Centered and User-Centered.*
As we mentioned, before the advantage of data centered is that scalability, with respect to the number of users, is better since a large number of users can be accommodated in the same logical channel. The logical channel is the set of resources needed to deliver one stream. These resources are needed both at the network and at the server to have a continuous delivery of streams. Moreover in data centered approach the bandwidth requirement is not so big because is proportional to number of objects (videos).

On the other hand a user centered algorithm dedicates channels in the side of individual users. Moreover user centered gives better control to the user over the playback of the video but the bandwidth required of by network increases proportionately with the number of users. The mode of service that user–centered uses is the client-server approach. After analyzing these two groups we can conclude saying that the best option would be to have something that has the positive properties of both, i.e. to have a mix of data and user centered. That is the so called hybrid approach.

In the following lines, we have a division about different groups of Broadcasting protocols that will be analysed through this thesis. There are four main groups that we divide into four chapters: the first one is the group that follows the Pyramid Broadcasting. Within it we find protocols that split the video into different segment sizes at equal bandwidth. The second group is the Harmonic Broadcasting protocol. In this group the video will be divided in equal segment sizes but at different bandwidth. In the third one we will have hybrid protocols, i.e. a mixture of Pyramid and Harmonic Broadcasting protocols, here the segment size and also the bandwidth will be equal although the frequency of the segment transmission will be different. The fourth group is the group of multicast protocols in which protocols are divided according to their disposition to follow a static multicast approach or a dynamic multicast approach.

Each group has different models that we will analyse deeply and in the next paper we will compare them getting the best options in each case.
I.5 **CONVENTIONAL BROADCASTING**

Conventional Broadcasting is the simplest Broadcasting method that exists but also the less efficient. What it simply does is to broadcast movies one after the other over the physical channel. This sequence is repeated several times. The viewer tunes into the channel and waits for her/his choice to be broadcast. When the movie starts Broadcasting it is downloaded in a secondary storage; thus, to download the video the user has to wait at most the access time $A$.

To get the access time we need the following parameters:

- $M = \#$ of movies
- $L =$ The length of the movie in time
- $B =$ nb (B=bandwidth, that is, $n$ times the consumption rate of a movie).  

$$a = \frac{ML}{n}$$

In figure 1 we can see how the typical and the simplest conventional Broadcasting looks like. In spite of its simplicity, this protocol presents several constraints that make it unpractical. As an example we will suppose that there are 10 movies of 2 hours each one with the consumption rate based in MPEG2 , i.e. $b = 5$Mb/s. We will suppose that the bandwidth is 500Mb/s The access time of the client to get the video will be $a = \frac{10 \times 7200}{100} = 720$ seconds or 12 minutes. Thus, we need to wait 12 minutes to watch a video. As we can observe this protocol is not worth because, with the increase of the number of videos, the waiting time would be relatively high. That is one of the lacks of the conventional broadcast that, as we will see through this thesis, can be solved with another protocols.

The second problem that we found is the necessity of a big storage capacity. If the transmission rate is bigger than the consumption rate of the movie and thus we have the data before we need, a secondary storage at the user end is necessary. To give a solution to this problem we have an **enhancement protocol**. This protocol partitions the channel into $z$ logical channels $C_1 \ldots C_z$. Each movie is broadcasted at its consumption rate over a fixed number of logical channels. They have a $d$ time units of delay between the successive broadcasts over consecutive channels. At each channel the same movie is broadcasted, i.e. a replica of each movie will be broadcast on the different logical channels. Thus, each channel
is broadcasted over L/d number of logical channels. The access time is given by \( a = d \) because a user has to wait for a maximum of \( d \) time units to start watching the movie. The bandwidth allocated for each movie will be \( B/M \). From equation (1) we know that \( B = nb \) and from equation (2) we know that \( A = ML/n \). Helped by this two equations we derived that \( \frac{B}{M} = \frac{L}{d}b \) and thus, \( d = \frac{ML}{n} \). Hence, the access time both at Conventional protocol and at Pyramid protocol is the same as we can observe:

\[
a = \frac{ML}{n}; \quad d = \frac{ML}{n}; \quad \text{thus, } a = d.
\]

The advantage of this enhancement protocol is that we do not need a secondary storage like in the Conventional protocol because the consumption rate of the movie and the broadcast bandwidth are the same. Figure 2 depicts how this protocol works.

![Figure 2: Enhancement protocol.](image)

In figure 2 we see how each video is broadcasted several times between different logical channels, all with a delay \( d \) from one to other.

On the other hand, as we explained in the section of data-centered vs user-centered, Conventional Broadcasting is a data-centered protocol so, for a fixed bandwidth of the channel, the access time becomes independent from the number of client but the longer is the cue of movies to be broadcasted the longer will also be the access time for the client because he/she has to wait to pass all the cue of movies until its turn arrives again. So, we can say that
this protocol is not scalable at all. We will see all the characteristics of this protocol on the table below:

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Conventional Broadcasting</th>
<th>Enhanced Conventional Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>Not scalable</td>
<td>Not scalable</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>Necessary</td>
<td>Not necessary</td>
</tr>
<tr>
<td><strong>Waiting time for the client</strong></td>
<td>$d = \frac{ML}{n}$</td>
<td>$d = \frac{ML}{n}$</td>
</tr>
<tr>
<td><strong>Server bandwidth</strong></td>
<td>TB</td>
<td>TB/M</td>
</tr>
</tbody>
</table>

*Table 2: Main characteristics of Conventional Broadcasting.*

In the following three chapters we will focus on classes of Video-On-Demand protocols that try to optimize on certain aspects of the basic Conventional Broadcasting protocol.
CHAPTER II

II.1 **BIASING THE SIZE OF SEGMENT**

In the previous chapter, we analysed the Conventional Broadcasting protocol. We explained how it works and which are its main features and deficiencies. We saw that although the simplicity of this protocol is not worth since the waiting time of the client is so high and a big storage capacity in the client size is necessary. Although it has an enhancement version that avoids the storage in the client size we concluded that it is not viable because for this we must have the same consumption rate at the server and at the client and furthermore the client access time is still so big. In the following chapter we explore a class of protocols that addresses the deficiencies that conventional Broadcasting has. We achieve these improvements by splitting each video into segments of increasing sizes and favouring the smallest segments Broadcasting them more times than the bigger ones. This helps in the following way: Each video is divided into segments of increasing size that are broadcasted into logical channels of equal bandwidth. In this way, the small segments will be broadcasted more times than those that are bigger and the access time of the client will be the time that the client need to receive the first segment.

In the following sections we will look into details some of the most popular representative protocols of this class of Video-On-Demand Broadcasting protocols so called: Pyramid
Broadcasting, Permutation based, Skyscraper and Fast Broadcasting Protocol. To ensure the continuous playback of the video the size of the data fragments must be chosen so that the playback duration of any fragment is longer than the worst latency in downloading the next fragment. To achieve low service latencies, the size of the first fragments can be made very small in order to allow them to be broadcasted more frequently. Although this approach is very efficient, it can only be used for very popular videos.

II.1.1 **PYRAMID BROADCASTING**

Pyramid Broadcasting [20], [21] was invented by S.Viswanathan and T.Imielinski in 1995. It is one of the pioneers in Broadcasting protocols for Video-On-Demand. The main characteristic of Pyramid Broadcasting protocol is that it achieves a significant reduction in the access time of the video. To reach this objective it takes advantage of the enormous difference between the available bandwidth of the network and the consumption rate of the video. We have to remark this point here because, as it can be remembered, Conventional Broadcasting considered that the transmission and consumption rate are equal. That assumption is wrong since it considers an ideal situation that will not be given. Thus, Pyramid Broadcasting finds a solution taking as a point of reference the real situation in which the client will have less bandwidth than the network. To take the advantage of this, it divides the physical channel into S logical channels of equal bandwidth. Each video is also divided in S segments, but these segments are of increasing size. Doing this we exploit the property of the bandwidth difference that we mentioned before. So, we multiplex the objects on the channels in such a way that the clients can start consuming the object as soon as possible.

This method favours small segments that are broadcasted several times more than those that are bigger. In this protocol the increase of the segments size reduces the frequency of Broadcasting that segments. In this way, we will get the reduction of the access time of the client because the time that the clients will need to start viewing the video will be the time that is needed to access the first segment. The mechanism works in the way that when the first segment is consumed the second one is already collected, when the second is consumed the third will be collected and then the fourth, fifth….and so on until the broadcast of the last
segment. With this process we will also ensure the continuity of the video because before the end of the consumption of segment $i$, the segment $i+1$ will be collected.

The algorithm that Pyramid Broadcasting follows is explained below; we will see it from the point of view of the server and the client.

In the server side the process works as follows:

1. The physical channel is divided into $S$ logical channels. Each segment has $B/S$ bandwidth.
2. Each object is divided also into $S$ segments. Each object (video) will be of size $L$. This size is measured in time units. Each segment has the a size of $L_i$ being $i = 1..S$.
3. $L_{i+1}$ is equal to $\alpha * L_i$ and $i = 1..S-1$.
4. The data segment $L_i$ will be broadcasted in channel $i$. We will do this with all the $L_i$ segments of all movies. Then in the same channel they will form a sequence being broadcasted periodically one after the other.

In the client side the following steps are involved:

1. Begin downloading the first data segment of the required object at the first occurrence and start consuming it concurrently.
2. Download data segments of size $L_i$ in order of $i = 1..S$ at the earliest possible time after beginning to consume data segment of size $L_{i-1}$ of the required object.
3. Any data segment of size $L_i$ is begun to download only from the start of that data segment.

The figure below depicts how this protocol looks like:

![Figure 3: Original Pyramid Broadcasting.](image)

As we can observe in the figure, it has a real pyramid form but in reverse in which the pieces that are on the top of the pyramid in this case are smaller than those that are in the base being broadcasted less times these ones. In the first channel we have all the first segments of all the videos broadcasted periodically several times. As we can appreciate, in the highest channels
each segment is broadcasted less times but their size is bigger than in the first one. The more we go through the pyramid the less will be the number of segments. Depending on the assumptions that we make we will have different results in the performance of the protocol. The result will not be the same for example when the bandwidth rate to broadcast the video is bigger than the consumption rate of the video or vice versa.

If both rates are equal it is not needed a secondary storage because as soon as the video segment is broadcasted will be consumed. On the other hand if the Broadcasting rate is bigger than the consumption rate of the video, then we need to store the video segments that arrive before we can use them. This storage device will be in the client side. In this way, we will have collected the segments that will be consumed later. In the worst case in Pyramid Broadcasting 2 segments will be downloaded at the same time; segments i and i + 1, and while the segments i is being consumed the segment i + 1 will be downloaded. A video of length L needs \( L * b \) Mb to be buffered. As we explained before the length of the movie is considered in seconds and the bit rate is in Mb/s. To calculate the minimum storage required to collected the segments we have to consider the last two segments of a video since they are the biggest as the Pyramid Broadcasting Protocol shows. So to calculate the buffer size needed we’ll do all the operations based on these two segments. We consider channels \( C_S \) and \( C_{S-1} \). These channels will have segments \( L_S \) and the segment \( L_{S-1} \). We will store \( L_{S-1} \) starting from

\[
(L_S - \frac{L_S}{B'}) * b \quad \text{and} \quad L_S \quad \text{starting from 1. In the worst case, as we mentioned in the previous page,}
\]

both segments will be downloaded in parallel. The time to consume segment \( (L_S/B') * b \) is \( L_S/B' \) and that is exactly the time when the information from the \( S^{th} \) channel is written to storage. When data from the \( S^{th} \) segment is almost completed to be written the data from \( L_S - 1 \) has already free positions from \( (L_S - \frac{L_S}{B'}) * b \) to \( L_S * b \). If \( L_S - 1 \) were written at a location preceding \( (L_S - \frac{L_S}{B'}) * b \) then \( L_S \) might overwrite some part of \( L_S - 1 \) before it is consumed. If \( L_S - 1 \) were written starting at location following \( (L_S - \frac{L_S}{B'}) * b \), then storage would be wasted.

Hence, we have to start writing \( L_S - 1 \) at location \( (L_S - \frac{L_S}{B'}) * b \) for the minimum storage and thus the storage must satisfy at least

\[
St \geq (L_S - \frac{L_S}{B'} + L_{S-1}) * b\
\]  
(3)
Another feature to take into account is the video continuity. Nobody wants to watch a video with jitters or hiccups. The client will not accept it, no way. To ensure the continuity of the video we have to consider the value of alpha. This value is derived from the continuity principle which assures the continuous view of the object.

The continuity principle says that Consumption Time \((L_i) \geq \text{Access Time} (L_{i+1}, B/S)\) where the Consumption Time \((L_i) = L_i\) and the Access Time instead (in return) denotes the upper bound time to begin downloading \(L_{i+1}\) on the \(i+1\) channel which has a bandwidth of \(B/S\), i.e. the access time says how much is the maximum time that we have to wait to start watching the video.

Since \(L_{i+1} = \alpha + L_i\) and \(\text{Access time}(L_{i+1}, \frac{B}{S}) = (L_{i+1} \cdot M)/\left(\frac{B}{S}\right)\) applying the continuity principle we derive that:

\[
\alpha \leq \frac{B}{S \cdot M}
\]  

From the equation above we also deduce that the required bandwidth to broadcast a video is

\[
TB = \alpha \cdot S
\]

The best improvement of Pyramid Broadcasting respect to the previous Conventional Broadcasting protocol is the access time of the client to start viewing the video. With Pyramid Broadcasting will show that this property is reduced notably. The user has not his/her video immediately. He/she has to wait some time. In Pyramid Broadcasting this time is the time that we need to broadcast the first segment. As we said before, in this protocol all segments are broadcasted one after the other in sequence so we do not have to wait more apart from the time that we need to receive the first segment. Furthermore, as we know, the first segment is the smallest and then we do not need so much time to access it. In the next lines we will prove how this work in a mathematical way.

The main characteristic of Pyramid Broadcasting is that it split the physical channel into logical channels. To calculate the access time we have to focus on the first logical channel. There we will have all the first segments of our \(M\) movies. To calculate this value we have to consider the worst case i.e. the case in which we miss the first segment of the movie and hence we have to wait until its next Broadcasting. Hence, we have to wait until all the
remaining first segments will pass until we get it again. The necessary time will be 
\[ T = \frac{L_1}{B'} \times M \]
Using Equation (3) that says that \( \alpha \leq \frac{B}{S} \times M \) then we deduce that:
\[ T = \frac{L_1}{\alpha} \] (6)
On the other hand, we know that for all channels from \( 2 \ldots S \), \( L_i = \alpha \times L_{i-1} \) or what is the same 
\( L_i = \alpha^{i-1} \times L_1 \). We also know that:
\[ L = L_1 + L_2 + L_3 + \ldots + L_s \] or what is the same using the term of \( \alpha \) that:
\[ L = L_1 \times (1 + \alpha + \ldots + \alpha^{K-1}) \] (7)
We can observe that it follows a geometrical growing, i.e. a geometrical serie thus with
\[ L = L_1 \times \frac{\alpha^S - 1}{\alpha - 1} \]
\[ L_1 = \frac{L \times (\alpha - 1)}{\alpha^S - 1} \] (8)
And substituying the 6th equation in the fourth we will get that 
\[ T = \frac{L \times (\alpha - 1)}{\alpha \times (\alpha^S - 1)} \] (9)
From the last equation, we can also conclude saying that \( \alpha \) must be \( \geq 1 \) because if not the access time will increase significantly.

The access time for the conventional Broadcasting is 
\[ T_C = \frac{ML}{n} \]
whereas for Pyramid Broadcasting is 
\[ T_P = \frac{L \times (\alpha - 1)}{\alpha \times (\alpha^S - 1)} \]
In the table below, we compare the access times of the protocols that we presented until now. 
We will see how this value improves in Pyramid Broadcasting. We will consider 10 movies 
of 2 hours = 7200 s and the bandwidth of the physical channel will be 100 times more than 
the consumption rate of the video. We will consider the optimal \( \alpha \) that is Euler’s constant, 2.72, as it will be proved later.
<table>
<thead>
<tr>
<th># of logical channels</th>
<th>Pyramid Broadcasting</th>
<th>Conventional Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>44.11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11.86</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4.03</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.41</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.19</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 3: Comparison between Pyramid Broadcasting and Conventional Broadcasting for 10 movies of 2 hours each.

If we look at table 3 we see the significant reduction of the access time that we get when we split the video into segments like Pyramid Broadcasting does. In Conventional Broadcasting in return we have just the physical channel, no segments, thus the access time is bigger. We can see this much better in the following graph. As we can observe, as more is the amount of segments more is the success of Pyramid Broadcasting. We can also realize about the bad performance of Pyramid Broadcasting if the number of segments is just one. In that case, it would be in the same conditions of Conventional Broadcasting and it is not worth. The bigger is the segmentation the better is the result with Pyramid Protocol.

Graph 1: Comparison of the access time of Pyramid and Conventional Broadcasting.
In the next table, we will calculate the Bandwidth that we need with a fixed access time. The bandwidth in each one is calculated in a different way but what all have in common is that we can calculate it having a fixed access time. What we are going to do now is exactly this, we will calculate the bandwidth with a fixed access time.

<table>
<thead>
<tr>
<th># of logical channels</th>
<th>Pyramid Broadcasting ( TB=\alpha*S )</th>
<th>Enhanced Conventional Broadcasting ( TB/M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.72</td>
<td>The same as the consumption rate</td>
</tr>
<tr>
<td>2</td>
<td>5.44</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8.16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10.88</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>13.6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>16.32</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4: Comparison of the bandwidth for Pyramid and Enhanced Conventional Broadcasting Protocols.*

Within these groups there are several protocols such as skyscraper, fast Broadcasting and permutation based, that differ in the way that they control the growth of segments. Skyscraper uses series and the rest on the other hand use geometrical progressions. So, skyscraper’s progression is lower and then the storage requirements at the client end will be greatly reduced.
Pyramid Broadcasting

It splits the video into segments of increasing size in channels of equal bandwidth

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>Scalable just for popular videos</td>
</tr>
<tr>
<td>Storage</td>
<td>$St \geq (L_S - \frac{L}{B'} + L_{S+1}) \cdot b$</td>
</tr>
</tbody>
</table>

- **Waiting time for the client**
  Reduced significantly respect to conventional Broadcasting. It is the time that we need to receive the first segment: $T_P = \frac{L \cdot (\alpha - 1)}{\alpha \cdot (\alpha^S - 1)}$

- **Minimum Bandwidth for a video**
  $TB = \alpha \cdot S$

Table 2: Main characteristics of Pyramid Broadcasting protocol.

### II.1.2 PERMUTATION BASED BROADCAST

Permutation -Based Broadcasting [26] was created in 1996 by Charu C.Aggarwal, Joel L.Wolf and Philip S.Yu. It is the protocol that solves the problems occurring in Pyramid Broadcasting. As we can remember, the main problem in Pyramid Broadcasting was the storage capacity. We need quite a lot of it. With Permutation based Broadcasting the storage and disk transfer are rather reduced. To solve this problem, what this new protocol does, is that instead of transmitting a segment in a very high bandwidth, it multiplexes each logical channel into P subchannels and transmits them at P times lower rate. The P substreams are staggered with each other to meet the same timing requirement. The latency also will be smaller. The segments increase geometrically. So as in the previous protocol the video will be partitioned into segments but then these segments will be again partitioned. This new partitions then will be transmitted in a cyclic permutation. And in this way the storage requirement will be reduced.
The way of operation of this protocol is so similar to Pyramid Broadcasting but its new procedure improves notably the storage necessities. The latency and the disk transfer rate also will have a clear improvement.

Before starting to explain how it works it is necessary to mention the assumptions that we made:

1. The first one is that a bit stream is periodic with period P if each bit i is identical to bit i+P.
2. Two identical periodic n bit streams X and Y have a phase difference $\Phi (-1 \leq \Phi \leq 1)$ if bit i in stream X is identical to bit $i + \Phi P$ in stream Y.
3. Stream X leaks or lags stream Y by a phase difference of $\Phi$ depending on the sign of the phase difference.
4. We shall create p periodic bit streams for each segment, where $p \geq 1$ is an integral parameter of the scheme.

As we did before with Pyramid Broadcasting protocol we will explain Permutation based protocol process from the server and client side.

From the server side it works like explained below:

1. Each logical channel is divided into $M_p$ subchannels having each of it a bandwidth of $\frac{B}{MPZ}$.

2. On channel i, each of the p subchannels corresponding to a given video will repeatedly broadcast the segment $S^{k}_{i}$.

Thus, these p subchannels will broadcast identical bit streams. But the bit stream on adjacent subchannels(modulo p) will have a phase difference of $1/p$.

We will analyse now how works this protocol in the client side. It follows in this way:

1. When the viewer requests the video k, the set top box at the client end latches onto the beginning of any copy of the segment $S^{k}_{i}$ the next time that is transmitted, and starts to consume this first segment.

2. In general, the client latches onto the first copy of the segment $S^{k}_{i+1}$ when the transmission of segment $S^{k}_{i}$ is completed.
The difference of Permutation Broadcasting respect to Pyramid Broadcasting is that there is no pipelining so we will have some latency between the end of the reception of one segment and the beginning of the reception of the next segment. During this latency we have to guarantee that there will not be any hiccup or interruption and that the video will continue; so we need to find the way to fill this space and assure the continuity. It is in this case when we have to remember about the existence of the buffer. The idea is to catch parts of the bit stream that we saved before in the buffer. It is for sure that we’ll have always parts of streams in the buffer because the transmission rate is always bigger than the display or consumption rate.

Figure 4 depicts how Permutation based Broadcasting works. In this case we suppose that there are 2 videos and we show the first segments of each one. The process works as follows:

In the first logical channel we will have \( p \) subchannels. In those subchannels the first segment of each video will be broadcasted. As we can observe in the first three subchannels we have broadcasted the first segments of the first video, in the next three channels we have broadcasted the first segments of the second video. Permutation based protocol follows.
Pyramid Broadcasting, i.e. segments are broadcasted in geometrical increasing size. Thus, in the second channel we will have the second segment of each video but the size of them will be bigger than in the first channel.

In this protocol, we will have a latency that was not in Pyramid Broadcasting. In Pyramid Broadcasting while we consume the segment i, the segment i+1 has been downloaded but in Permutation based Broadcasting protocol we’ll finish the display of the segment before to start downloading segment i+1 so we will have a latency between both segments. This is why we use a buffer to store parts of previous segments as we explained above; because we need to occupy this vacuum. As we are mentioning through all the thesis, the most important characteristics when we analyse Video-On-Demand protocols are the access time for the client, the bandwidth and the storage requirement. In the following lines we will explain how we measure them in Permutation Based Broadcasting.

The access time of the client is crucial to decide if the protocol is good or not. In this case we must say, that respect to Pyramid Broadcasting, this protocol improves it in a notably way. As in the previous protocol, in Permutation based Broadcasting the access time of the client will be the access time to get the first segment. We should always consider the worst case, i.e. when we miss the first segment and we have to wait for it until is broadcasted again. In the previous case if we loose the first segment we have to wait until the first segments of the rest of the videos are broadcasted but in this new protocol we wait to the next segment, i.e. the first segment is broadcasted in p subchannels all the time and in each subchannel there is a phase difference of 1/p from one to other. In this way the access time of the client will be:

$$T = \frac{1}{p} \frac{\frac{1}{TB} \frac{1}{L}}{pMZ}$$

Permutation based protocol also follows like Pyramid Broadcasting protocol the geometrical progression of segment increasing size. In this way we know that \(L_{i+1} = \alpha L_i\) and so

$$L_i = \frac{L \cdot (\alpha - 1)}{\alpha \cdot (\alpha^S - 1)}.$$

Applying this we get that

$$T = \frac{1}{p} \frac{\frac{L(\alpha - 1)}{TB} \frac{1}{L}}{pMZ} = \frac{(\alpha - 1) \cdot Z}{(\alpha^S - 1) \cdot TB}$$

Now that we have the required access time for Permutation Based Pyramid Broadcasting (PBB) we’ll depict in the table below which are the numeral differences with Pyramid Broadcasting (PB).
Having the formulas of the access time:

\[ T_{PBB} = \frac{(\alpha-1)Z}{(\alpha^S-1)TB} \]  \text{ and }  \[ T_{PB} = \frac{L* (\alpha-1)}{\alpha * (\alpha^S-1)} \].

Again, we will consider the optimal alpha to look at the differences in the best case. Like before we have \( M = 10 \) movies and \( L = 2h = 7200 \) s. the total bandwidth will be like before 100 times more than the consumption rate of the video. We have to take into account that the # of channels is not the same in both protocols. If we have 4 channels in Pyramid Broadcasting we should realize about the fact that in Permutation based Broadcasting this channels will be split into more channels so the number of channels will be bigger than in PB. So, first of all, we will explain how is done the calculation of the number of the subchannels. We have to consider the value \( p \). We have \( S \) channels in Pyramid Broadcasting and we know that in PBB each channel \( S \) is split in \( p \). So the total number of channels in PBB will be \( S \cdot p \). In this case, we will consider that \( p \) is 4.

<table>
<thead>
<tr>
<th># of logical channels (PB)</th>
<th>Pyramid Broadcasting</th>
<th># of logical channels (PBB)</th>
<th>Permutation Based Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>44.11</td>
<td>4</td>
<td>( 128 \times 10^{-3} )</td>
</tr>
<tr>
<td>2</td>
<td>11.86</td>
<td>8</td>
<td>( 4.59 \times 10^{-5} )</td>
</tr>
<tr>
<td>3</td>
<td>4.03</td>
<td>12</td>
<td>( 1.25 \times 10^{-6} )</td>
</tr>
<tr>
<td>4</td>
<td>1.41</td>
<td>16</td>
<td>( 2.29 \times 10^{-8} )</td>
</tr>
<tr>
<td>5</td>
<td>0.51</td>
<td>20</td>
<td>( 7.001 \times 10^{-10} )</td>
</tr>
<tr>
<td>6</td>
<td>0.19</td>
<td>24</td>
<td>( 1.5 \times 10^{-11} )</td>
</tr>
</tbody>
</table>

*Table 3: Comparison between Pyramid and Permutation Based Broadcasting protocols.*
Graph 2: Representation of the access time between Pyramid and Permutation Based Broadcasting protocols.

In graph 2 we represent table 3 to see the access time requires for each protocol and we can observe how the new protocol improves notably this aspect.

Now we will be centered in the calculation of the bandwidth. The smaller the calculation is the better it will be. The storage requirements will be the last point to be analysed in this protocol. We know that in the previous protocol the storage requirements are significant. The necessity of the storage is substantial. Furthermore, we showed that the quantity is relatively big. In Permutation Based protocol this lack of Pyramid Broadcasting is rather decreased.

Another important issue is the storage capacity. We remarked before that Pyramid Broadcasting protocol needs rather much storage. To finish with the analysis of Permutation Based we will explain how is calculated the storage that it needs.

We assume that in the worst case the total transmission time $t$ of video $i$ is

$$T = \frac{L_i}{TB} \cdot \frac{ZMP_{TB}}{L_i} = \frac{L_i}{TB} \cdot \frac{MSp}{TB}$$  \hspace{1cm} (11)$$

The amount of video consumed in time $t$ is equal to $tC$. Hence, the total storage is $L_i - tC$. Using this and and the equation (11) we get, the total storage is

$$St = L_i - L_i \cdot \frac{MSp}{TB}$$  \hspace{1cm} (12)$$
On the other hand, to calculate the total bandwidth that this protocol needs we assume from [26] that \( p \leq \frac{TB}{MSb} - \alpha \). Hence, the total bandwidth will be

\[
TB = MSb(p+\alpha) \tag{13}
\]

<table>
<thead>
<tr>
<th># of logical channels (^{(PB)})</th>
<th>Pyramid Broadcasting (B=\alpha*S)</th>
<th># of logical channels (^{(PBB)})</th>
<th>Permutation Based Broadcasting (TB=MSb(p+\alpha))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.72</td>
<td>4</td>
<td>18.96</td>
</tr>
<tr>
<td>2</td>
<td>5.44</td>
<td>8</td>
<td>37.92</td>
</tr>
<tr>
<td>3</td>
<td>8.16</td>
<td>12</td>
<td>56.88</td>
</tr>
<tr>
<td>4</td>
<td>10.88</td>
<td>16</td>
<td>75.84</td>
</tr>
<tr>
<td>5</td>
<td>13.6</td>
<td>20</td>
<td>94.8</td>
</tr>
<tr>
<td>6</td>
<td>16.32</td>
<td>24</td>
<td>113.76</td>
</tr>
</tbody>
</table>

*Table 4: Comparison of bandwidth between Permutation Based and Pyramid Broadcasting protocols.*

In the table above, we have the comparison between Pyramid and Permutation based protocols. As we can see, the bandwidth that Permutation based need is more than Pyramid Broadcasting method. To make easier the calculation of the bandwidth in Permutation based we took into account the following formula that is proportional to the original one that we got before: \( Z(p+\alpha) \).
In the graph above, we depicted the required bandwidth for Pyramid Broadcasting protocol and Permutation Based Broadcasting protocol and we see that although Permutation Based improves the storage requirement it needs more bandwidth.

To finish with the whole analysis of Permutation Based protocol below we have a table that shows all the main characteristics that it has.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>Yes, for popular videos</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>$St = L_i - L_iZ_{Spb/TB}$</td>
</tr>
<tr>
<td><strong>Waiting time for the client</strong></td>
<td>$T_{PBB} = \frac{(\alpha - 1) \cdot Z}{(\alpha^S - 1) \cdot TB}$</td>
</tr>
<tr>
<td><strong>Minimum Bandwidth for a video</strong></td>
<td>$TB = MSb(\alpha + \alpha)$</td>
</tr>
</tbody>
</table>

*Table 4: Main characteristics of Permutation Based Broadcasting protocol.*
II.1.4 **SKYSCRAPER**

Skyscraper was invented in 1997 by Kien A.Hua and Simon Sheu[4]. It is a protocol within Pyramid Broadcasting and it improves the lacks that has Permutation Based Broadcasting. The problem of Pyramid Broadcasting is that although it offers good access latency it needs a big storage capacity and a big disk bandwidth at the client end. With Permutation based Broadcasting this problems were solved but a new problem appeared; the access latency increased and the synchronization is more difficult. So, Skyscraper Broadcasting was invented to find a solution to these problems.

Skyscraper proposed a new data fragmentation scheme and also a new Broadcasting strategy. We will prove how improves the lacks that the previous ones had. This protocol divides the video into segments of increasing sizes like Pyramid Broadcasting and Permutation based Broadcasting but it differs that it a geometrical growth. It uses a serie. Like in the previous two protocols the bandwidth of each channel will be equal.

We divide the bandwidth (TB) into \([TB/b]\) logical channels of b Mbits/s each. These channels are allocated among the M videos such that there are \(K= \left[\frac{TB}{bM}\right]\) channels for each video.

In order to broadcast a video over its S dedicated channels, each video file is partitioned into S segments using the data segmentation scheme. Each of these segments is broadcasted repeatedly on its dedicated channel at its consumption rate b. The beginning of the next segment must be encountered before the current segment is consumed.

As we mentioned in the previous paragraph the difference in respect to the rest of the protocols analysed until now is that here we use a serie progression instead of geometric progressions \([1, a, a^2, a^3, \ldots]\). To generate them we use the following function:

\[
F(n) = \begin{cases} 
1 & n = 1 \\
2 & n = 2,3 \\
2.f(n - 1) + 1 & n \mod 4 = 0 \\
f(n - 1) & n \mod 4 = 1 \\
2.f(n - 1) + 2 & n \mod 4 = 2 \\
f(n - 1) & n \mod 4 = 3
\end{cases}
\]
or

\[
F(n) = \begin{cases} 
1 & \text{if } n = 1 \\
2 & \text{if } n = 2,3 \\
(2 + 2\left\lfloor \frac{n}{2} \right\rfloor - n)f(n-1) + (1 + 2\left\lfloor \frac{n}{2} \right\rfloor - n)(1 + \left\lfloor \frac{n-4}{4} \right\rfloor) & \text{otherwise}
\end{cases}
\]

The series looks as follows: [1, 2, 5, 12, 25, 25, 12, 12, K, K, ..., K]:

- K = width of the skyscraper
- [2, 2] = even group
- [25, 25] = odd group

The first number of the series means that the size of the first segment is one unit, i.e., \( L_1 \). The size of the second one is 2 units, so, \( 2\times L_1 \). The third also will be 2 units, the forth 5 units… and so on. K is the width of the skyscraper and we use it to restrict segments from becoming too large. The biggest size of a segment will be K, because the buffer space will be bigger if we let to be bigger K. The width of the skyscraper can be controlled in such way that we get the desired access latency. The number of videos determines S. With S we adjust K and then we limit the size of \( L_1 \).

The advantages of this technique are several, such as the unlimited scalability, the exponential reduction of the service latency with the increase in the server bandwidth and since K segments are downloaded sequentially, buffer requirement is minimal.

The process that follows is simple. There are 2 loaders: the odd and the even. Each one downloads even and odd groups. The S segments are downloaded sequentially using one loader. So, when the buffer is full of these groups then the video player will use data from there. The bigger is \( L_1 \) the smaller will be W. In broadcast protocols one of the goals is to reduce the access latency and we know that the access latency is \( L_1 \) so using a big K we will reduce \( L_1 \) and as a consequence also the access latency. The access latency is defined as follows:

\[
\text{Access Latency} = \frac{L}{S \sum_{i=1}^{\min(f(i), K)}} \quad (11)
\]
The server multiplexes among the S*M logical channels. Each of the channels broadcasts repeatedly one of the S*M data segments. We will form groups; odd and even groups. It works in this way: having the series that we showed above \([1,2,2,5,5,12,12,25,25,\ldots K, K, \ldots K]\), we’ll form different groups. The first group \([q]\), the second group \([2,2]\), the third will be \([12,12]\) and so forth. These groups interleave with each other, being one odd one even, one odd, one even and so on. They are played in the order that they occur.

We mentioned before the usage of a buffer. An important aspect to analyse is the storage capacity that we need. To analyse it, it is important to let it clear the classification of transition groups. We have three kinds of segments groups they will be the odd group, the even group and the first group. They are represented in this way:

1. \([1]\) \([2,2]\): this is the first group. It happens just once, at the beginning of the transmission.
2. \([A,A]\) \([2A+1, 2A+1]\): This is the even transition. For example from \([2,2]\) to \([5,5]\)
3. \([A,A]\) \([2A+2, 2A+2]\): This is the odd transition. For example from \([5,5]\) to \([12,12]\)

We will denote \(T_S\) as start time of the video. When \(T_S\) is odd the client doesn’t need to buffer data because it will be played as soon as arrives but when \(T_S\) is even the client have to start to prefetch the second group as soon as it begins to playback the first group at time \(T_S\). At time \(T+2\) it must start to preload the second half of group 2 while is playing back the first half of the same group, then it will play the second half of the second group when it preloads the first part of the third group and so on until we play all the video. This process is represented below:

\[\text{Figure 4 : Odd Version of Skyscraper.}\]
In skyscraper the video playback will be without hiccups because all time will have prefetched data in the buffer.

The bandwidth that Skyscraper needs is equal to the number of logical channels that it needs,

\[ B = \mathbb{Z} \]  \hspace*{1cm} (12)

The storage capacity that it requires will be calculate in the following way: we know that the last segments are the biggest ones. So assuming that we have the transition

\[ [A, A] ^{2A+1, 2A+1} \]

and the last group transition of the series \((X, X) \rightarrow (K, K, \ldots, K)\) applying to the serie that transition \([4]\) we will get the storage capacity required

\[ S_T = bL_1(T_{SK} - 1) \]  \hspace*{1cm} (13)
To sum up we summarized all parameters that we have calculated during the analysis of Permutation Based Broadcasting in the next table:

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>No limit</td>
</tr>
<tr>
<td>Storage</td>
<td>$S_T = bL_1(T_{SK} - 1)$</td>
</tr>
</tbody>
</table>

Waiting time for the client

$$T_{SK} = L_1 = \sum_{i=1}^{S} \min(f(i), K)$$

Minimum Bandwidth for a video

$$B = Z$$

Table 5: Main characteristics of Skyscraper protocol.

II.1.5 **FAST BROADCASTING**

Fast Broadcasting, Li-shen Juhn and Li-ming Tseng 1997 [27], is the last that was invented within Pyramid Broadcasting. It is the most efficient compared with its predecessors. It needs less bandwidth and at the client end the buffer requirement is between previous pyramid schemes. Moreover, if there is no buffer the client still can have his/her movie but with a waiting time. However, with previous protocols we must have a buffer.

Fast Broadcasting improves clearly the aspect of bandwidth requirement but although it can maintain the continuity of the video without the buffer, the buffer requirement is bigger than in permutation based broadcast; something that with previous protocols was impossible.

It is the most efficient in this group. The video is divided into geometrical series and the bandwidth is equal to the consumption rate. The client has to download from all K streams at the same time.

The process that follows in the server side is the following:

1. We have C channels for each movie of length L. The bandwidth of each one will be equal to the consumption rate of the video.
2. Each video will be divided into $S$ segments. The division follows a geometrical serie.

$$S = \sum_{i=0}^{C-1} 2^i = 2^C - 1$$  \hspace{1cm} (13)

3. As in the protocols described before $L = L_1 + L_2 + L_3 + \ldots + L_S$. In each channel $C_i$ we will have $2^i$ segments where $i = 0..C - 1$, $j = 2^i$ and $k = 2j - 1$ and they will be broadcasted periodically.

In the client end, the process works in this way:

1. The client has to download from all streams at the same time. The process starts downloading the first segment $S_1$ in channel $C_0$ and concurrently the rest of the segments from the rest of the channels.
2. As soon as we start receiving the data segments we can start viewing it.
3. We will stop the process in channel $C_i$ when we have received $2^i$ data segments from this channel.

<table>
<thead>
<tr>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
<th>$S1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S2$</td>
<td>$S3$</td>
<td>$S2$</td>
<td>$S3$</td>
<td>$S2$</td>
<td>$S3$</td>
<td>$S2$</td>
<td>$S3$</td>
<td>$S2$</td>
</tr>
<tr>
<td>$S4$</td>
<td>$S5$</td>
<td>$S6$</td>
<td>$S7$</td>
<td>$S4$</td>
<td>$S5$</td>
<td>$S6$</td>
<td>$S7$</td>
<td>$S4$</td>
</tr>
<tr>
<td>$S8$</td>
<td>$S9$</td>
<td>$S10$</td>
<td>$S11$</td>
<td>$S12$</td>
<td>$S13$</td>
<td>$S14$</td>
<td>$S15$</td>
<td>$S8$</td>
</tr>
</tbody>
</table>

*Figure 6: Scheme of Fast Broadcasting protocol for 4 channels.*

To see all the process exactly in its detail we have to pay attention in the picture above. As we explained before, all segments start to download concurrently. So, when we start with $S_1$, $S_2$ will arrive at the same time or just as soon as $S_1$ has been consumed. We will define the time needed to consume a segment as $t$. We will load the segment $S_1$ at $t_0$, so if we want to download the segments from channel $C_i$ then these segments will arrive at time $t_0 + (2^i - 1)*t$.

Another important point is that to start consuming segments from $C_i$ we must had consumed all the previous ones.

As it is being mentioned through all the document, the necessity of the buffer is a point that deserves analysis. In this case, when the transmission rate of the segments is bigger than that of consumption rate then we will need to storage the segments that are arriving before the previous segment is consumed. To analyse the buffer requirements we have to consider the
last channel because the maximum size of the buffer will be when there are the segments from this channels in.

In channel $C_i$ we will have $2^i$ segments. So, in the last channel $C_{c-1}$ there will be $2^{c-1}$ segments. All $S$ segments of the video need $(2^{c-1}) \cdot t$ time to arrive to the buffer and in that time we will be consumed just $2^{c-1}$ segments so the rest, i.e. $S - 2^{c-1}$ must stay in the buffer.

Being $b$ the consumption rate the size of the segment will be $L \cdot b / S$.

So having $S - 2^{c-1}$ and $L \cdot b / S$ we will conclude saying that the size of the buffer must be.

\[
S_T = (S - 2^{c-1}) \cdot \frac{Lb}{S} = \frac{C-1}{2} - 1 \cdot Db
\]

We will arrive to this formula using equation (14)

and thus at the end we get

\[
\frac{2^{C-1}}{2} - 1 \cdot Db < \frac{Db}{2}
\]

The access time that the client has to wait until he/she can start viewing the video is the time that we need to get the first segment taking into account the worst case that is when we miss it.

Hence, having the video of length $L$ divided into equal $S$ segments we deduce that the required access time for the client is

\[
T_F = \frac{L}{S} = \frac{L}{2^{c-1}}
\]

The last point to analyse is the bandwidth requirement. We will see that the required bandwidth of Fast Broadcasting is equal to the Permutation Based’s. According to the storage requirements the maximum I/O rate requirements will occur during the interval $[t_0 + T_F, t_0 + 2T_F]$. As we have to take into account the worst case we should consider the case when we want to read the second segment but we have also to write the input data from the second channel to the last channel. Hence, and knowing all this the bandwidth rate for Fast Broadcasting will be

\[
TB = Z \cdot b
\]
We remarked before that one of the novelties of Fast Broadcasting is that we can continue viewing the video without any discontinuity although in the absence of a buffer. We will conclude saying that the bigger is the buffer the less will be the waiting time.

<table>
<thead>
<tr>
<th>Fast Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>It divides the video into segments of equal size that follow the geometrical progression having a bandwidth equal to the consumption rate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>No limit</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>$S_T = \frac{C-1}{C} - \frac{1}{Db}$</td>
</tr>
<tr>
<td><strong>Waiting time for the client</strong></td>
<td>$T_F = \frac{L}{S} = \frac{L}{2^C - 1}$</td>
</tr>
<tr>
<td><strong>Minimum Bandwidth for a video</strong></td>
<td>$TB = Z*b$</td>
</tr>
</tbody>
</table>

*Table 6: Main characteristics of Fast Broadcasting protocol.*

**II.2 COMPARISON BETWEEN DIFFERENT MODELS**

In the current chapter we have analysed the different protocols that split their videos into segments of increasing sizes. We have seen that all this protocols divide the physical channel into logical channels of equal bandwidth. As they are data centered protocols when the number of users increase the network will not be exhausted. On the other hand we analysed the protocols taking into account a reduce number of videos. The reason is that in this way we see how each protocols results when popular videos are broadcasted.

In the previous paragraph we saw that the scalability of these protocols is not limited because we made the assumption which says that for popular videos during a time interval during the day we will have enough bandwidth to broadcast the videos.

We saw that the Pyramid Broadcasting protocols was a good idea when the only protocol know until that date was the Conventional Broadcasting protocol. As we could observe the reduction of access time that it supposed was quite high. The lack of this protocol is the
storage capacity that needs at the client end. Because of it Permutation Based protocol was created. This protocol reduces rather the storage capacity thanks that it splits each logical channel into P subchannels. They will have P times lower rate and will be transmitted in a cyclic permutation that is why the storage is reduced notably. We can observe that also in the formulas. As we can see in Permutation Based protocol there is a division with the total bandwidth whereas in Pyramid there is not:

\[
St(\text{Permutation based}) = L_i - \frac{L_iZ_Sbp}{TB} \quad St(\text{Pyramid}) \geq (L_S^t - \frac{L}{B'} + L_{S+1})*b
\]

The problem of Permutation based Broadcasting protocol is that although the storage capacity is reduced there is no pipelining like in Pyramid Broadcasting protocol. Hence, there is latency between the end of the reception of one segment and the beginning of the reception of the next one. To solve this problem we will use a buffer. All this process makes Permutation Based Broadcasting to have a difficult synchronization and higher latency.

The third protocol that we have presented is Skyscraper. It was published as a solution for Permutation Based Broadcasting. Skyscraper also divides the video into segments of increasing sizes but the difference respect to the precious protocols is that it follows a serie progression instead of the geometrical.

To finish with this section we have spoken about Fast Broadcasting. It is the newest protocol of this group. It is also the most efficient compared with its predecessors. It needs less bandwidth and at the client end the buffer requirement is between previous pyramid schemes. Moreover, if there is no buffer the client still can have his/her movie but with a waiting time. However, with previous protocols we must have a buffer.

Fast Broadcasting improves clearly the aspect of bandwidth requirement but the buffer requirement is bigger than in permutation based broadcast although it can maintain the continuity of the video without the buffer, something that with previous protocols was impossible.
CHAPTER III

III.1 BIASING THE BANDWIDTH OF SEGMENT

In the first chapter we have seen protocols for Video-On-Demand that follow a Pyramid schedule. In the previous chapter we have seen how each video is divided into different segments that are allocated into logical channels of equal bandwidth. We have analysed these protocols and we have observed how they work and the advantages and disadvantages that each one has. We saw for example that the simplest and easiest Conventional Broadcasting protocols is not worth because the big storage capacity that needs and the long access time that offer for the client. On the other hand, we saw that Pyramid Broadcasting solves this lacks although the storage capacity still is rather big. Each protocol that we have presented tries to solve the lacks of the previous one but none of them gives the optimal solution yet. In the second chapter we present a new family called Harmonic Broadcasting. Protocols that follow Harmonic Broadcasting have the following feature in common. All these protocols split the video into equal segment sizes but at different bandwidth rates, i.e. in this way we will see how the bandwidth will be notably reduced although we will also see the lacks like for example the delay of data delivering. That is why new protocols appeared years later each one trying to improve the original Harmonic Broadcasting. In the following section we will analyse in depth this protocol and show all the process that it follows. In this chapter we will see them deeply and we will compare the necessary access time, the bandwidth and the
storage requirement that they need and in the second section of this chapter we will compare them with the protocols of the previous chapter to see the positive and negative sizes of each of them.

### III.1.1 HARMONIC BROADCASTING

Harmonic Broadcasting [28], [29] was invented in 1997 by Juhn and Tseng. Comparing with the protocols that we analysed before it gets the lowest bandwidth to achieve the given access time. But its problem is that it not always delivers data on time. In 1997 Juhn gave a solution to this problem proposing a delay at the beginning of the consumption of the video. But while this method fixes one problem, it creates another one. The access time increases significantly and so, Harmonic Broadcasting looses all its advantages.

Harmonic Broadcasting splits the video into segments of equal size. Each segment is broadcasted repeatedly on its own stream and the customer must receive all streams at once. Segment $S_i$ will be broadcasted in channel $C_i$ with bandwidth $b/i$. The sum of the channel bandwidths will be the following:

$$\sum_{i=1}^{Z} b = b \sum_{i=1}^{Z} \frac{1}{i} = bH(Z),$$

where $H(Z)$ is the Harmonic number of $n$. (18)

H(n) grows slowly so the bandwidth will be rather smaller than the consumption rate. In Harmonic Broadcasting we will speak about time slots. A time slot is the time that the user needs to consume just a segment. We will see later that it is also the maximum waiting time for a client to receive a segment:

$$T_{SL} = \frac{S_1}{b}.$$  (19)

In this protocol the client has to wait for the beginning of an instance of segment $S_1$ before it can start receiving data. As we mentioned before once that the client starts receiving $S_1$ he/she will receive all the other streams. Before continuing with how works the process of
Harmonic Broadcasting we have to define another concept that needs a definition: the subsegment. It is the part of the segment that the client receives during a timeslot.

The first segment will have just a subsegment because the transmission and consumption rate are the same. The rest will be divided into i equal subsegments: $S_{i,1}, S_{i,2}, S_{i,3}, \ldots, S_{i,i}$. Segment two will have two subsegments, segment three will have three subsegments, segment four has 4 subsegments and so on. We will need a buffer at the client because we have to collect the segments that we receive before they are need.

The goal of HB is that when the client is ready to consume segment $S_i$, it have already received data for i-1 slots of time. So the client will have i-1 from i subsegments in the buffer and it will receive the last subsegment while it is consuming the full segment.

There are two problems in Harmonic Broadcasting. The first one is the data delay as we have been telling through all the report. And the second one is that it requires the client to wait $(z-1)d/z$. We cannot make the client wait any time because doing that what at the end we do is to make increase the video access time and so, reduce all the advantages that make Harmonic Broadcasting to be better respect to other protocols.

To illustrate better all the explanations given until this point in the next paragraphs we explain the step by step:

1. Start downloading the first data segment ($S_1$) of the required video on the first channel $C_1$ and download concurrently the rest of the segments in their respective channels ($C_2, \ldots, C_z$).
2. Right after we begin to download the segments we can start to consume the movie with its normal speed in order.
3. We will stop downloading segments when we have received all the i subsegments of $S_i$ from that channel. The i - 1 subsegments of $S_i$ will be written in the buffer and consumed concurrently when the last subsegment is received.
If we have a look to figure 7 we can appreciate how the bandwidth of channels is always less as we increase their number, i.e. in the first channel we will have b, in the second b/2, in the third b/3 and in the fourth b/4. In this way, the first segments will arrive sooner than the last ones.

![Figure 7: Original Harmonic Broadcasting.](image)

If we have a look to figure 7 we can appreciate how the bandwidth of channels is always less as we increase their number, i.e. in the first channel we will have b, in the second b/2, in the third b/3 and in the fourth b/4. In this way, the first segments will arrive sooner than the last ones.

![Figure 8: Another representation of Original Harmonic Broadcasting.](image)

Figure 8 represents also the Harmonic Broadcasting protocol. We can observe that the bandwidth is smaller as the as the number of channel increase. We also see in his figure how each segment is divided into subsegments of equal size. And how the # of subsegments in each channel coincides with the channel number.

On the other hand, on the figure bellow depicts the size of the segments and also the time slot.
Stream 1:

\[ \begin{array}{cccc}
S_1 & S_1 & S_1 & S_1 \\
\end{array} \]

Stream 2:

\[ \begin{array}{cccc}
S_{2,1} & S_{2,2} & S_{2,1} & S_{2,2} \\
\end{array} \]

Stream 3:

\[ \begin{array}{cccc}
S_{3,1} & S_{3,2} & S_{3,3} & S_{3,1} \\
\end{array} \]

Figure 9: Representation of the Harmonic Broadcasting protocol for the first three streams.

Looking at the graph above, we will prove the data delay of this protocol. We will call \( t_0 \) at the time that the client will start receiving the second instance of segment \( S_1 \). Consider this moment.

At time \( t_0 + d \) the client will be ready to consume the segment \( S_2 \) and it will be ready to consume segment \( S_2 \) and it will have one subsegment from the segment \( S_{2,2} \) in its buffer. However, it will require all of the data from subsegment \( S_{2,1} \) by time \( t_0 + 3d/2 \) but it will not receive it until time \( t_0 + 2d \). So, to fix this problem of the time delay the simplest solution is to make the client before starting consuming data, but we cannot make it wait any time because in that case HB will lose its efficiency. We should make the minimum time. In the next lines will see how much must wait.

One of the main features in Video-On-Demand is the bandwidth as we are reminding throughout the thesis. In Harmonic Broadcasting the calculation of this parameter is simple. We know that the movie is divided into \( Z \) segments and each segment is divided in the same way into subsegments. On the other hand, we know that the bandwidth of each channel is \( b/i \). So with all this information the total bandwidth needed for the video is

\[
TB = \sum_{i=1}^{Z} \frac{b}{i} = H^*b, \quad \text{where} \quad H = \sum_{j=1}^{Z} \frac{1}{j}
\]

Another characteristic that we are analysing through the thesis is the storage capacity; how to minimize it. In this case, the transmission rate will be also greater than the consumption rate.
First of all, we have to clarify all parameters that we use. We have \( Z \geq 2 \), i.e. that the number of channels is at least 2. On the other hand, we have \( t_0 \). This parameter expresses the time in which we begin to load \( S_1 \) from channel 1. So, during the time interval \([t_0+(i-1)d,t_0+i*d]\) the subsegments coming from \( C_{i+1..C_Z} \) must be buffered. The next step to calculate the necessary storage size is to calculate how much increased data size we need. That will be written as below:

\[
I_i = \frac{S_i}{N} \sum_{j=i+1}^{Z} \frac{1}{j}, \text{where } 1 \leq i < Z \tag{20}
\]

In the same time interval the client consumes the segments that he/she has received before. The reason is that it cannot download and play the segment concurrently. So, the previous \( i-1 \) subsegments of segment \( S_i \) are consumed with the last received subsegment of \( S_i \) from channel \( C_i \). So, we have to calculate the output data that are read from the buffer between the interval \([t_0+(i-1)d,t_0+i*d]\). The equation that follows is the next:

\[
O_i = \frac{S_i}{Z} \frac{i-1}{i}, \text{where } 1 \leq i \leq Z \tag{21}
\]

On the other hand, there is the parameter of the buffer size that will be \( BS \). \( BS_i \) will be the buffer size required at time \( t_0+i*d \). At time \( t_0+d \) all data that come from \( C_2 \) to \( C_Z \) need to be buffered. The buffer size needed for segment one will be \( Z_1=I_1 \) because it has not previous ones. But for the rest of the segments the buffer size will be:

\[
BS_i = BS_{i-1}+I_i-O_i, \text{where } 1 < i < Z \tag{22}
\]

During \( t_0+(Z-1)d \) to \( t_0+Zd \) the last received subsegment of \( S_Z \) will be consumed with other \( Z-1 \) subsegments of \( S_Z \). There is no write requirement and all buffered data will be consumed during the last interval. Hence we have \( I_Z = 0 \) and \( BS_Z = 0 \).

The last point to analyse is the access time of the client. To calculate it we need the bandwidth. The problem of this group is that data are not delivered always in time. In the following paragraphs we show how much access time has to wait the user and further on we will see two new protocols that resolve this problem. They are Cautious-Harmonic, Quasi-Harmonic and Polyharmonic Broadcasting protocols. We defined before the subsegment as a part of the segment and we know that a time slot is the time needed to receive a subsegment.
So, the waiting time for the client will be the time that we need to receive the subsegment, thus the time slot:

\[ T_{SL} = \frac{S_1}{b} = \frac{L}{S} \]  

(23)

And hence, we deduce also that the maximum waiting time that the client has to wait is also the same.

As we can observe is quite big the time that the client has to wait to start viewing the video. The bandwidth requirement reduces notably but at the expense of the access time. We will show in the next three protocols how this can be improved.

<table>
<thead>
<tr>
<th>Harmonic Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>It divides the video into segments of equal size but different bandwidth</td>
</tr>
<tr>
<td><strong>Characteristics</strong></td>
</tr>
<tr>
<td>Scalability</td>
</tr>
<tr>
<td>Storage</td>
</tr>
<tr>
<td>Waiting time for the client</td>
</tr>
<tr>
<td>Minimum Bandwidth for a video</td>
</tr>
</tbody>
</table>

*Table 6: Main characteristics of Harmonic Broadcasting protocol.*

**III.1.2 CAUTIOUS-HARMONIC:**

Cautious-Harmonic protocol was invented by Jehan-François Paris, Steven W.Carter and Darrell D.E.Long in 1998. It is the solution for the data delay. It follows the philosophy of Harmonic Broadcasting protocol but solves the problem appeared with the data delay.

The video is partitioned into \( n \) equally size segments. The first channel repeatedly broadcast the first segment \( S_1 \) at the playback rate. The second channel alternately broadcasts \( S_2 \) and \( S_3 \)
also repeatedly but at half the playback rate. They alternate the full bandwidth. The rest of segments \((S_i)\) are repeatedly broadcasted on its dedicated channel at a rate of \(b_i = b/i\). With this technique, the client will receive a segment at a full bandwidth or will have the whole segment before it is needed. In this protocol, the playback will start when the first stream can be downloaded. In the first three segments we will not have any delay because the first three segments are consumed at a full bandwidth and in the rest of the segments neither because the entire contents of segment \(i\) will be retransmitted every \(i-1\) subsegments and will thus be available to client before it starts consuming \(S_i\). The bandwidth that this protocol enhancement needs is the following:

\[
TB_{\text{CHB}} = 2b + \sum_{i=1}^{Z-1} \frac{b}{i} = \frac{b}{2} + bH(Z-1) \tag{24}
\]

\[
TB_{\text{HB}} = \sum_{i=1}^{Z} \frac{b}{i} = H_Z * b , \text{where } H_Z = \sum_{j=1}^{Z} \frac{1}{j}
\]

We can see the difference that there is between them. The Harmonic number of one is calculated by \(H(Z-1)\) and the other is \(H(Z)\). In this way, we can observe how the original Harmonic Broadcasting method needs less bandwidth than Cautious Broadcasting protocol.

In this protocol the waiting time for the client will be the the same of the minimum waiting time for Harmonic Broadcasting:

\[
T_{\text{CH}} = \frac{1}{Z} \tag{25}
\]

This protocol is better, than for example, skyscraper in terms of server bandwidth but on the other hand requires more client bandwidth than it.

<table>
<thead>
<tr>
<th><strong>Cautious-Harmonic Broadcasting</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The method is the same as in Harmonic Broadcasting protocol but in this case there is not data delay</td>
<td></td>
</tr>
<tr>
<td><strong>Characteristics</strong></td>
<td><strong>Summary</strong></td>
</tr>
<tr>
<td>Scalability</td>
<td>No limit</td>
</tr>
<tr>
<td>Waiting time for the client</td>
<td>(T_{\text{CH}} = \frac{1}{Z})</td>
</tr>
<tr>
<td>Minimum Bandwidth for a video</td>
<td>(TB_{\text{CHB}} = 2b + \sum_{i=1}^{Z-1} \frac{b}{i} = \frac{b}{2} + bH(Z-1))</td>
</tr>
</tbody>
</table>

*Table 7: Main characteristics of Cautious-Harmonic Broadcasting protocol.*
III.1.3 QUASI-HARMONIC

Quasi-Harmonic protocol is the best option within Harmonic Broadcasting protocols. It was invented by the authors of Cautious-Harmonic Broadcasting. Both protocols appeared in 1998. As we have seen new protocols try to improve the previous ones and in this case, Quasi-Harmonic Broadcasting protocol also does.

It assures all the subsegments arriving at the client end before the client starts to consume the segment. It divides each segment into $m$ equal subsegments and broadcasts the first segment repeatedly on the first channel. The difference in this protocol is that each segment $i$ from $1 < i \leq Z$, is partitioned into $im-1$ fragments for some parameter $m$, and the client will receive $m$ fragments from each channel per timeslot. In the same way, each time slot is divided into $m$ equally subsegments and so, the client will receive one fragment during each subsegment. The secret of this protocol is that the fragments are not broadcasted in order. The last subslot of each subslot is used to broadcast the first $i-1$ fragments repeatedly, and the rest are ordered such that the $k^{th}$ subslot of slot $j$ is used to broadcast fragment $ik+j-1 \mod i(m-1)+i$. Having this mapping we are able to compute the bandwidth requirement of Quasi-Harmonic Broadcasting protocol.

The problem of these two protocols is that they have less bandwidth efficiency. There is a third protocol called Polyharmonic Broadcasting that is similar to these ones but it differs from downloading the video in the moment that the clients want, i.e. the segment cannot downloaded from the beginning but anyway it will be joined to form a whole segment and so the segment will be downloaded before the previous is finished playing.

The main problem of Harmonic Broadcasting is the delivery on time as we recall in the lines above. With the third group we will solve this problem because there is the same fixed delay for all customers that want to watch the same video. Thus, the transmission of all segments is reduced. As a result, the total bandwidth can be also reduced. This one is the only protocol that has all conditions for an efficient Broadcasting protocol. The secret of this protocol remains in the organization of the fragments. We have to take into account the channel $i$. The last subslot of each timeslot is used to transmit the first $i-1$ fragments of $S_i$ in order. The rest
of the subslots transmit the other i(m-1) fragments such that the k\textsuperscript{th} subslot of slot j is used to transmit fragment (ik+j-1)mod i(m-1)+i. This protocol will be explained in the next section.

In the following graph we will see how is the process of Quasi-Harmonic Broadcasting protocol:

<table>
<thead>
<tr>
<th>S1</th>
<th>S1</th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_{2,2}</td>
<td>S_{2,4}</td>
<td>S_{2,1}</td>
</tr>
<tr>
<td>S_{3,2}</td>
<td>S_{3,6}</td>
<td>S_{3,1}</td>
</tr>
</tbody>
</table>

*Figure 10: Division that Quasi-Harmonic Broadcasting protocol follows.*

To clarify the figure that we have above in the next lines we show all the operations. In the first row of the figure where we have the sequence S\textsubscript{1}, S\textsubscript{1}, S\textsubscript{1} we did not follow any formula; we have just follow the protocol that say that the first segment will be broadcasted repeatedly all time on the first channel. For the rest we used the formulas that are above.

To remind the notation:

- i= channel
- k=sub slot
- j=timeslot

We choose m=3

The second row is built as follows:

- i=2; k=1; j=1; (2*2+1+1-1) mod 2(3-1)+2 = 2 mod 6 = 2
- i=2; k=2; j=1; (2*2+1+1) mod 2(3-1)+2 = 4 mod 6 = 4
- i=2; k=3; j=1; (2*3+3+1-1) mod 2(3-1)+2 = 6 mod 6 = 0
- i=2; k=1; j=2; (2*1+2+1-1) mod 2(3-1)+2 = 3 mod 6 = 3
- i=2; k=2; j=2; (2*2+2+1) mod 2(3-1)+2 = 5 mod 6 = 5
- i=2; k=3; j=2; (2*3+2+1) mod 2(3-1)+2 = 7 mod 6 = 1
- i=2; k=1; j=3; (2*1+3+1-1) mod 2(3-1)+2 = 4 mod 6 = 4
- i=2; k=2; j=3; (2*2+3+1) mod 2(3-1)+2 = 6 mod 6 = 0
- i=2; k=3; j=3; (2*3+3+1) mod 2(3-1)+2 = 8 mod 6 = 2

Note that we refer to first subsegment when we have the results 0 and 1.
The third row is built as follows:

\[
\begin{align*}
  i &= 3; \ k = 1; \ j = 1; \ (3*1+1-1) \mod 3(3-1)+3 = 3 \\
  i &= 3; \ k = 2; \ j = 1; \ (3*2+1-1) \mod 3(3-1)+3 = 6 \\
  i &= 3; \ k = 3; \ j = 1; \ (3*3+1-1) \mod 3(3-1)+3 = 0 \\
  i &= 3; \ k = 1; \ j = 2; \ (3*1+2-1) \mod 3(3-1)+3 = 4 \\
  i &= 3; \ k = 2; \ j = 2; \ (3*2+2-1) \mod 3(3-1)+3 = 7 \\
  i &= 3; \ k = 3; \ j = 2; \ (3*3+2-1) \mod 3(3-1)+3 = 1 \\
  i &= 3; \ k = 1; \ j = 3; \ (3*1+3-1) \mod 3(3-1)+3 = 5 \\
  i &= 3; \ k = 2; \ j = 3; \ (3*2+3-1) \mod 3(3-1)+3 = 8 \\
  i &= 3; \ k = 3; \ j = 3; \ (3*3+3-1) \mod 3(3-1)+3 = 2
\end{align*}
\]

Until now, we have seen how is the process that Quasi-Harmonic Broadcasting protocol follows. Now we will analyse the main issues that we are analysing throughout thesis.

As we see, Quasi-Harmonic Broadcasting divided each segment into \( i m - 1 \) subsegments to be broadcasted over in subslots whereas Harmonic Broadcasting assigns one subsegment to each subslot. Thus, Quasi-Harmonic Broadcasting uses the remaining subslots to broadcast a redundant copy of the first subsegment of each segment to guarantee that the consumer will always have the first \( i - 1 \) subsegments of each segment before it starts consuming data from that segment. Assuming this redundancy, we know that now we will broadcast more information in the same time interval and hence the bandwidth will increase:

\[
TB = b + \sum_{i=2}^{Z} \frac{bm}{im-1} = bH(Z) + \sum_{i=2}^{Z} \frac{b}{i(im-1)}
\]  \hspace{1cm} (26)

The last point is the access time. In Quasi-Harmonic Broadcasting is the same as the minimum waiting time for Harmonic Broadcasting protocol. Hence the access time will be:

\[
T_{CH} = \frac{1}{Z}
\]  \hspace{1cm} (27)
Quasi-Harmonic Broadcasting

It is the best option within Harmonic protocols. Offers less bandwidth and access time. The novelty is the way of fragmentation

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>No limit</td>
</tr>
<tr>
<td>Waiting time for the client</td>
<td>$T_{CH} = \frac{1}{Z}$</td>
</tr>
<tr>
<td>Minimum Bandwidth for a video</td>
<td>$TB = b + \sum_{i=2}^{Z} \frac{bm}{im-1} = bH(Z) + \sum_{i=2}^{Z} \frac{b}{i(im-1)}$</td>
</tr>
</tbody>
</table>

Table 8: Main characteristics of Quasi-Harmonic Broadcasting protocol.

### III.1.4 POLYHARMONIC BROADCASTING

Polyharmonic Broadcasting protocol [14] was invented by Jehan-François Pâris, Steven W.Carter and Darrel D.E.Long in 1998. It is the latest version in Harmonic Broadcasting protocols family. We will see how this new protocol achieved less bandwidth for the same access time.

Polyharmonic Broadcasting protocols has two new differences to the previous Harmonic protocols. The first one is that it requires that the set top box starts downloading data from the moment a customer requests a specific video instead of waiting until the customer begins watching the beginning of the first segment. The second difference is that makes all customers to wait the same waiting time without looking when that made the request.

The process that follows is the one in Harmonic protocols, i.e. it divides the video into segments of equal sizes. $l = \frac{L}{Z}$. the protocol will assign $Z$ channels for these $Z$ segments. Each stream $I$ will repeatedly show segment $S_i$. In Polyharmonic Broadcasting protocol no client can start consuming the first segment of the video before having downloaded data from all $s$ streams during the time interval $t = ml$ where $m$ is an integer $\geq 1$. With this what we get is that the segment is not consumed until $(m+i-1)l$ time units passed. So retransmitting the segment at a rate $b_i = \frac{b}{m+i-1}$ we will assure that all the contents of segment $I$ will be in set top box before the client starts viewing the video.
We can now deduce which will be the total bandwidth that this protocol needs.

\[
TB = \sum_{i=1}^Z b_i = b \sum_{i=1}^Z \frac{1}{m + i + 1} = b(H(Z + m - 1) - H(m - 1))
\]  

(28)

In Polyharmonic Broadcasting protocol we do not have to wait until the first segment is downloaded. So, the maximum waiting time that the client has to wait to view the video will be also the minimum waiting time. Knowing that \( t = ml \) and \( l = L/Z \) we can deduce that

\[
T_{PH} = t = \frac{L}{k}
\]  

(29)

where \( k \) is the number that express the \# of segments how many times is multiple of \( m \).

On the other hand, in this we can derive that keeping \( k \) constant and increasing \( m \) and \( Z \) the bandwidth will be lower, i.e. we will see how the bandwidth of Polyharmonic Broadcasting protocol is smaller than the bandwidth of the original Harmonic Broadcasting protocol

\[
TB(k, m+1) \leq TB(k, m)
\]

Furthermore what the author of Polyharmonic Broadcasting protocol do is to compute the limit of \( TB(k, m) \) when \( m \) and \( Z \) goes to infinite and \( k \) remains constant. In this way, they get the lower bandwidth.

\[
\lim_{m \to \infty} TB(Z, m) = \lim_{m \to \infty} \sum_{i=1}^Z \frac{b}{m + i + 1} = \int_0^L \frac{b}{T_{PH} + t} dt = \log \frac{T_{PH} + L}{T_{PH}} = \log(k + 1)
\]

(30)

Through all the thesis we are analysing the most three important features that make a Video-On-Demand protocol to be Good or bad. At this point we have to analyse the storage requirements that Polyharmonic Broadcasting needs. We have to consider the quantity of data that we can receive during a time slot. Depending on the time slot that we are we will calculate the amount of data differently.
To find the storage requirement that Polyharmonic Broadcasting protocol needs we have to compute also the data consumed during a time slot. In the equation below, we see that depending in the factor of $m$ will change it:

$$R_i = \begin{cases} 
\frac{db \sum_{j=1}^{Z} 1}{m + j + 1} & 1 \leq i \leq m \\
\frac{db \sum_{j=i-m+1}^{Z} 1}{m + j + 1} & m < i \leq Z + m - 1 \\
0 & i = Z + m 
\end{cases}$$

(31)

We have already calculated the two main parameters that we need to get the storage capacity: the data receive during a time slot and the data that consumes during it. Having this two parameters and the data that the client has in the buffer after each time slot we can compute

$$S_T = S_{T_{i-1}} + R_i - C_i$$

(33)

where $S_{T_{i-1}}$ is the data that the client has after the time slot $i-1$. Hence, the maximum $S_T$ will be the storage required.
In the graph below, we will see how Polyharmonic Broadcasting protocol looks like:

**Server Stream 1:**

\[
\begin{array}{cccccc}
S_{1,1} & S_{1,2} & S_{1,1} & S_{1,2} & S_{1,1} & S_{1,2}
\end{array}
\]

**Server Stream 2:**

\[
\begin{array}{cccccc}
S_{2,1} & S_{2,2} & S_{2,3} & S_{2,1} & S_{2,2} & S_{2,3}
\end{array}
\]

**Server Stream 3:**

\[
\begin{array}{cccccc}
S_{3,1} & S_{3,2} & S_{3,3} & S_{3,4} & S_{3,1} & S_{3,2}
\end{array}
\]

**Client:**

\[
\begin{array}{ccc}
delay
\end{array}
\]

\[
\begin{array}{ccc}
S_1 & S_2 & S_3
\end{array}
\]

*Figure 11: The process of Polyharmonic Broadcasting protocol for 3 streams.*

To resume all that we have said the following table shows a summary of Polyharmonic Broadcasting protocol:

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>No limit</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>( S_T = S_{T_{i-1}} + R_i - C_i )</td>
</tr>
<tr>
<td><strong>Waiting time for the client</strong></td>
<td>( T_{PH} = t = \frac{L}{k} )</td>
</tr>
<tr>
<td><strong>Minimum Bandwidth for a video</strong></td>
<td>( TB = b(H(Z+m-1) - H(m-1)) )</td>
</tr>
</tbody>
</table>

*Table 9: Main characteristics of Polyharmonic Broadcasting protocol.*
III.2 **Comparison Between Different Models**

Harmonic Broadcasting protocol is a new family within data centered Broadcasting protocols. It presents a different way of Broadcasting. Instead of dividing the video into segments of increasing sizes it splits them into segments of equal sizes thus the reduction of the bandwidth is notable. The problem is that the original Harmonic Broadcasting method does not deliver all data in time and hence loses the advantage that gets with the bandwidth.

The original Harmonic Broadcasting broadcasts each segment repeatedly on its own stream and the customer must receive all streams at once. There are two new protocols that improve the original protocol. We have seen the Cautious-Harmonic and the Quasi-Harmonic Broadcasting protocols. The first one is an enhancement of the original Harmonic protocol and the second one is the best solution.

With Cautious-Harmonic Broadcasting protocol the problem that Harmonic Broadcasting protocol has is solved. To achieve it partitions the video into n equally size segments. The first channel repeatedly broadcast the first segment $S_1$ at the playback rate. The second channel alternately broadcasts $S_2$ and $S_3$ also repeatedly but at half the playback rate. They alternate the full bandwidth. The rest of segments ($S_i$) are repeatedly broadcasted on its dedicated channel at a rate of $b_i = b/i$.

But although it fixes the problem of data delay we could also observe that for Cautious-Harmonic Broadcasting the bandwidth requirement is bigger:

$$TB_{CHB} = 2b + \sum_{i=3}^{z-1} \frac{b}{i} + \frac{bH(Z-1)}{2} \quad (34)$$

$$TB_{HB} = \sum_{i=1}^{z} \frac{b}{i} = H_z * b$$

Because of the increase of bandwidth that Cautious-Harmonic protocol supposes we have seen that the same authors created Quasi-Harmonic Broadcasting protocol. The change that it makes respect to Cautious-Harmonic protocols is that in this protocol each segment $i$ from is partitioned into im-1 fragments for some parameter m, and the client will receive m fragments from each channel per timeslot. In the same way, each time slot is divided into m equally subsegments and so, the client will receive one fragment during each subsegment. The secret
of this protocol is that the fragments are not broadcasted in order. Thanks to this, the bandwidth is again reduced but still is quite high.

In the last section of this chapter, we have analysed the Polyharmonic Broadcasting Protocol. It is the best option within the Harmonic Broadcasting protocols. It is the newest protocol. We have remarked the two novelties that it has. The first one is that it requires the set top box to start downloading data from the moment a customer requests a specific video instead of waiting until the customer begins watching the beginning of the first segment. The second difference is that makes all customers to wait the same waiting time without looking when that made the request. Hence, the bandwidth that it requires is the following:

$$TB = \sum_{i=1}^{Z} b_i = b \sum_{i=1}^{Z} \frac{1}{m + i + 1} = b(H(Z + m - 1) - H(m - 1)) \quad (35)$$

On the other hand, we calculated also the access time for the client and storage requirements for him/her and we have seen that

$$T_{PH(\text{Polyharmonic})} = t = \frac{L}{k} \quad (36)$$

As we can observe the waiting time of the formula is the same as in the Harmonic Broadcasting protocol but the difference is that for Polyharmonic Broadcasting protocol it will be also the maximum waiting time while for the Harmonic Broadcasting protocol not.

$$T_{SL(\text{Harmonic})} = \frac{S_l}{b} \quad (37)$$

As a last point to analyse, we calculated the storage capacity that the best protocol of the group of Harmonic Broadcasting protocol needs. We have calculated that the storage capacity that it requires is the data that we have storaged until the previous time slot plus the quantity of data that we can receive during the current time slot minus the data that are consumed during the same time slot:

$$S_l = ST_{i-1} + R_i - C_i \quad (38)$$
CHAPTER IV

IV.1 BIASING THE FREQUENCY OF THE SEGMENT TRANSMISSION

In the fourth chapter we will treat protocols that bias the frequency of the segment transmission. These protocols split the video into segments of equal size and broadcasted them with the same bandwidth but at different frequency. Until now it is the best option presented in Video-On-Demand Broadcasting. The protocols presented in this section are the called Pagoda Broadcasting protocols. We will present the Original Pagoda Broadcasting protocol and the Fixed Delay Pagoda Broadcasting method. This new family of protocols is the result of mixing the Pyramid Broadcasting and the Harmonic Broadcasting protocols. It takes the advantages of these two Broadcasting methods. It will split segments into equal sizes like in Harmonic Broadcasting because in that way it gets lower bandwidth requirements and like in Pyramid Broadcasting they will be broadcasted with equal bandwidth but the novelty is that segments will be broadcasted at different periodicities.
IV.1.1 PAGODA BROADCASTING PROTOCOL (THE HYBRID)

This protocol was invented by Paris, Crater and Long in 1997 [10]. It is the result of mixing Pyramid and Harmonic protocols. Within this group we have three kinds of Broadcasting methods. The first one is the original Pagoda Broadcasting, the second one is the New Pagoda Broadcasting and the third is the Fixed delay Pagoda Broadcasting. Each of it improves the previous protocol. Like in Harmonic protocols, Pagoda Broadcasting divides the video into a number of equal size segments. Each segment has the same duration. This duration is called timeslot. The timeslot is the time that the client needs to consume a single segment of the video as we mentioned in the previous protocol. Like in pyramid Broadcasting, it broadcasts these segments at the same bandwidth but with different periodicities. The problem is that it is not easy to choose a proper segment to stream mapping and the proper Broadcasting periodicity for each segment. This protocol achieves to have one stream for each segment thanks to the time domain multiplexing among many segments sharing a few streams. In Pagoda Broadcasting each channel broadcasts data at the playback rate and client receives data from all channels simultaneously. The playback will start as soon as it can download the first segment.

The process that Pagoda Broadcasting follows is the one explained below: we have to remark that the process starts from channel 2:

1. Divide the video into n segments where n will change depending on k. If k is even then n will be $4(5^{k/4})-1$ and if it is odd will be $2(5^{k/2})-1$. Start with a even stream $E= 2^j$
2. Start with a even stream $E= 2^j$
3. Denote $r$ the index of the lowest-numbered segments $S_r$ transmitted by $E$
4. $S_r$ transmitted at $1/rd$
5. Consider $z$ contiguous slots within stream $2^j$. All odd slots will be allocated to segments $r$ to $\frac{3r}{2}-1$ and all will be broadcasted at $\frac{1}{rd}$. The rest will be broadcasted at $\frac{1}{2rd}$ and will be allocated from $2r$ to $3r-1$. 
To see better how it really works we have bellow a figure of the first three channels.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$ to $S_{3r/2-1}$</td>
<td>2k</td>
</tr>
<tr>
<td>$S_{3r/2}$ to $S_{2r-1}$</td>
<td>2k+1</td>
</tr>
<tr>
<td>$S_{2r}$ to $S_{3r-1}$</td>
<td>2k</td>
</tr>
<tr>
<td>$S_{3r}$ to $S_{5r/2-1}$</td>
<td>2k+1</td>
</tr>
</tbody>
</table>

**Figure 11: Pagoda Broadcasting process for 4 streams.**

1. The first stream transmits segment $S_1$ at frequency $1/d$. So all time it will be broadcasting $S_1$ in channel 1.
2. Stream 2 differs a little bit: The odd slots contain segment $S_2$ so it will be transmitted at frequency $1/2d$. The even slots alternate between segments $S_4$ and $S_5$ and the frequency will be $1/4d$.
3. Stream 3 will be as follows. It will be transmitted at $1/3d$. The rest will be transmitted at $1/6d$: $S_6$, $S_7$, $S_8$, $S_9$.
4. Stream 4 will be as follows. It will be transmitted at $1/4d$ and the rest at $1/8d$.

<table>
<thead>
<tr>
<th>Pagoda Broadcasting protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Segments</strong></td>
</tr>
<tr>
<td>$S_1$ to $S_{3r/2-1}$</td>
</tr>
<tr>
<td>$S_{3r/2}$ to $S_{2r-1}$</td>
</tr>
<tr>
<td>$S_{2r}$ to $S_{3r-1}$</td>
</tr>
<tr>
<td>$S_{3r}$ to $S_{5r/2-1}$</td>
</tr>
</tbody>
</table>

**Table 10: Segment to channel mapping of Pyramid Broadcasting.**

In *table 7* we see how in Pagoda Broadcasting are organized the segments. Which segments will be in each stream and the frequency at they will be broadcasted. The way that it uses to calculate the number of segments that it will be allocated in each channel follows the next formula:

$$ z(Z) = \begin{cases} 
4(5^k-1) - 1 & Z = 2k \\
2(5^k) - 1 & Z = 2k + 1 
\end{cases} $$

(39)
The main advantage of it is that the server bandwidth is low compared to Skyscraper. But in the other side the client bandwidth is much higher. Another disadvantage is that is expensive. Pagoda Broadcasting protocol has an enhancement protocol called New Pagoda Broadcasting. It improves the original one but the client requirement bandwidth is still high. Another issue that we are analysing in this thesis is the storage requirement. The maximum storage that we need will be when the Set-Top-Box starts downloading less data that it is consuming. Imagine that we have an even number of streams or logical channels. Thus \( E = 2^j \). Since the last stream will contain over half of the segments of the video the Set-Top-Box will download at least as much data as it consumes the entire time it is downloading data. Since we also know that when \( E \) is even we have \( 2(E^{k-1}) \).

**IV.1.2 NEW PAGODA BROADCASTING PROTOCOL**

New Pagoda Broadcasting protocol [11] appeared in 1999, i.e. two years after of the first Pagoda Broadcasting protocols by the hand of the same authors. We saw before how Pagoda Broadcasting works, how is the segment to channel mapping, its bandwidth, storage requirement, access time etc. The first novelty that New Pagoda Broadcasting protocol presents is the way of doing the segment to channel mapping.

The New Pagoda Broadcasting protocol uses the column major order instead of the row major order like the Original Broadcasting protocol uses. Hence, it will look as it follows:

![Figure 12: Division of Pagoda Broadcasting using column major order.](image)

If we compare with row major mapping as we can see it below:

![Figure 13: Division of New Pagoda Broadcasting using row major order.](image)
Observing the two figures above we see that thanks of the new structure only the segments from 10 to 15 are broadcasted once every 10 slots like in the original one. The rest will be broadcasted every 15 slots and furthermore we can add two new segments because of the save of bandwidth. In this new case instead of concentrating into pairs we will concentrate in all available slots. Hence, we will ensure that each segment \( i \) is broadcasted once every \( i \) slot. Thus, knowing that this is the maximum number \( S_{\text{max}} \) of segments that could be transmitted using \( m \) streams of bandwidth \( b \) will have to satisfy the inequality

\[
\sum_{i=1}^{S_{\text{max}}} \frac{b}{i} \leq Sb
\]  

(40)

No segment to block mapping would allow any variant of our protocol to use less bandwidth than the Quasi-Harmonic Broadcasting protocol to guarantee a given maximum viewing delay.

### IV.1.3 FIXED-DELAY PAGODA BROADCASTING

Fixed-Delay Pagoda Broadcasting (Jehan François Pâris, 2001) is the best between all Pagoda protocols presented until now. It is a protocol that makes a user to wait for a small delay for watching the video that he/she has requested. The goal of this delay is to reduce the bandwidth requirement.

We have to consider a video of length \( L \) and \( k \) channels. The bandwidth of each channel is equal to the consumption rate of the video. As a characteristic of all Pagoda protocols we have to divide the video into \( n \) equal-size segments of duration \( d = L / \text{number of segments} \). The difference of Fixed-Delay Pagoda Broadcasting with respect to others is that the client has to wait for a fixed time to watch the video. The notation used will be \( w = md \) where \( m \) is an integer equal or bigger than 1, \( m \geq 1 \). In this way the first segment, \( S_1 \), will be transmitted at least once every \( m \) slots to be always received before the customer starts watching it. To generalize every segment \( S_i \) will be transmitted at least once every \( m+i-1 \) slots.
The Fixed-Delay protocol partitions each channel $C_i$ into $S_i$ subchannels in such a way that slot $i$ of channel $i$ belongs to its subchannel $i \pmod{S_i}$. Each subchannel has thus $1/S_i$ of the slots and $1/S_i$ of the bandwidth of channel $C_i$.

We remarked before that in Pagoda Broadcasting protocols is very important the way in we do the segment to channel mapping; depending on this a part of its performance. The Fixed-Delay Broadcasting protocol does it in a strict sequential way. The process that follows is the explained below:

1. The first segments of the video are mapped into subchannel 0 of channel $C_1$.
2. The next segments of the video are mapped into subchannel 1 of channel $C_1$ and so on until all $S_1$ subchannels of $C_1$ are used.

This process repeats for the subchannels from channel $C_2$ to $C_k$. Hence, the whole segment-to-channel mapping can be derived from its $k+1$ parameters, that are

1. The number $k$ of channels allocated to the video.
2. The ratio $m$ between the customer waiting time and the segment duration $d$.
3. The numbers $S_1$, $S_2$, $S_k$ of subchannels for each of the $k$ channels.

The optimal number of subchannels for a channel depends on the periodicity at which the segments assigned to this channel had to be retransmitted.

The bandwidth of this protocol is calculated in the following way. Assuming that we have the consumption rate $b$ and the small interval $\Delta t$ at a location $t$ within the video and assuming that each customer Set-Top-Box starts downloading video data from the moment the video is ordered, the data of this time interval will have to be broadcasted at a minimum bandwidth of

$$\frac{b}{t + w}$$

(41)

Having equation 41 if we consider the limit when $\Delta t$ goes to 0, we will get the minimum bandwidth that is needed to broadcast the video, that is the following

$$B_{\text{min}} = \int_0^L \frac{b}{t + w} dt = \log \left( \frac{L + w}{w} \right)$$

(42)
From this equation we are able to derive also the minimum waiting time. When the transmission bandwidth is equal to $k$ times the consumption rate of the video the waiting time will be

$$T_{FDP} = \frac{L}{e^k - 1}$$

(43)

To sum up, with Broadcasting protocols it is needed to say that service latency can be improved by increasing bandwidth in 2 ways: the first one is increasing the client bandwidth like Harmonic and Pagoda do and the second one is increasing the server bandwidth like Skyscraper does.

<table>
<thead>
<tr>
<th>Fixed-Delay Pagoda Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segments of equal size broadcasted at equal bandwidth but different periodicities.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalarity</strong></td>
<td>No limit</td>
</tr>
<tr>
<td><strong>Waiting time for the client</strong></td>
<td>$T_{FDP} = \frac{L}{e^k - 1}$</td>
</tr>
<tr>
<td><strong>Minimum Bandwidth for a video</strong></td>
<td>$B_{min} = \int_{0}^{L} \frac{b}{(t+w)} dt = \log \frac{L+w}{w}$</td>
</tr>
</tbody>
</table>

*Table 11: Main characteristics of Fixed-Delay Pagoda Broadcasting protocol.*

### IV.2 COMPARISON BETWEEN DIFFERENT MODELS

This group is the best option of all data centered Broadcasting methods. It is the optimal solution because it combines the advantages of the group of Pyramid Broadcasting with the advantages of the group of Harmonic Broadcasting. It will split segments into equal sizes like in Harmonic Broadcasting because in that way it gets lower bandwidth requirements and like in Pyramid Broadcasting they will be broadcasted with equal bandwidth but the novelty is that segments will be broadcasted at different periodicities.
To do the segment to channel mapping the original Pagoda Broadcasting protocol uses the row column order while the New Pagoda Broadcasting protocol uses the major column order. In this way, the New Pagoda Broadcasting protocol can have more segments for each stream because of the saving that makes in the bandwidth.

The difference of Fixed-Delay Pagoda Broadcasting respect to other two Pagoda protocols is that the client has to wait for a fixed time to watch the video. Inserting this novelty to the basic process that Pagoda protocol follow we will reduce the access time for the client apart from the bandwidth as we mentioned before. In the following table we show a comparison among the three Pagoda Broadcasting protocols. In this table we can see from oldest protocols until the newest the access time improves.

<table>
<thead>
<tr>
<th>Broadcasting</th>
<th>Fixed-Delay Pagoda Broadcasting</th>
<th>New Pagoda Broadcasting</th>
<th>Original Pagoda Broadcasting</th>
</tr>
</thead>
<tbody>
<tr>
<td>number &amp; number of channels</td>
<td># of segments</td>
<td>Maximum waiting time</td>
<td># of segments</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>0.75</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>0.2143</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>116</td>
<td>0.07759</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>292</td>
<td>0.03082</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>770</td>
<td>0.01169</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>2046</td>
<td>0.004399</td>
<td>172</td>
</tr>
<tr>
<td>7</td>
<td>5477</td>
<td>0.001643</td>
<td>442</td>
</tr>
</tbody>
</table>

*Table 12: Comparison of access time between Fixed-Delay Pagoda Broadcasting, New pagoda Broadcasting and Pagoda Broadcasting.*

In the table above, from the first moment the access time reduction in Fixed Delay Pagoda Broadcasting is of 25% respect to New Pagoda and Pagoda Broadcasting protocols. We can also observe in the high proportion in which the number of segments increase. The bigger is the number of segments the less will be the access time for the client and in this case the bigger is the number of segments in less channels the better will be the result.
Graph 4: Representation of the access time for Fixed-Delay, New Pagoda and the Original Pagoda Broadcasting protocols.

In the graph above, we can observe the difference between the access time of the three Pagoda protocols. We can observe how Fixed-Delay Pagoda Broadcasting is the best option from the beginning giving the less access time for the client.
Section II: User-centered protocols

CHAPTER V

In the fifth chapter we have a new kind of protocols; a new different family. Different from the protocols presented until this point. As we have seen up to now all protocols were a data centered based protocols. At this point of the thesis we will present a serie of protocols that works completely different respect to the rest. They are protocols centered in the user. As we have explained in the first chapter a user centered algorithm dedicates channels in the side of individual users. Moreover user centered gives better control to the user over the playback of the video but the bandwidth required by network increases proportionately with the number of users. The mode of service that user–centered uses is the client-server approach. After analyzing these two groups we can conclude saying that the best option would be to have something that has the positive properties of both, i.e. to have a mix of data and user centered. That is the called hybrid approach.

As mentioned in the above paragraph the best solution is to have a hybrid between data and user centered services. Users’ requests are collected over a period of time and are then collectively allocated in a logical channel. Most of the protocols belong to this approach. We will consider the next three ones: Batching, Patching and Piggybacking.

All the Broadcasting protocols that we will see are based into a server but the user centered group also can be based into peer to peer. In this way the server will serve the video to less users and the users will be responsible of passing the video to other users. Users will be
sharing the videos. Using peer to peer we will notably reduce the bandwidth requirement and also the access time for the client.

In this chapter we will split protocols into static multicast approach and dynamic multicast approach. In this chapter we will analyse the topic of Broadcasting versus multicasting. Broadcasting protocols are efficient for popular videos. They need a predetermined server bandwidth to achieve certain access latency and they are not subject to data loss. But this kind of protocol is not flexible in providing True or interactive VOD service. To get this goal multicast is presented keeping the server bandwidth requirement low.

V.1 **STATIC MULTICAST APPROACH**

In static multicast a video server serves a batch of requests for the same video that arrive within a short period of time using one server channel. That is why in this group we have the Batching protocol. All clients of the same batch receive the same data from the same multicast tree. Within Batching there are different schemes. The difference among them is that they use a different method to decide which batch to serve first. These kinds of protocols are Near-Video-on-Demand because users have to wait in a queue to get the video data.

V.1.1 **BATCHING**

It is known that Video-On-Demand consumes a lot of bandwidth that is why one of the goals of all protocols is to reduce the server bandwidth. To do this it is sent out a single stream to multiple clients using IP multicast. Batching protocols can be selected in two ways. The first one is Batching by size and second one is Batching by timeout. We will analyse the second one but both of them will be explained in the following lines.

There are two kinds of Batching:

- **Batching by size**: It follows a Poisson equation [40]. Firstly the batching size and the arrival rate of requests for the object start must be predefined. When batching size number of requests are accumulated then I/O stream will be initiated. This
policy reduces the I/O demand on the storage server but the disadvantage is that we can have long delays for requests.

- **Batching by timeout:** The first step is to assign a timer. This is done when a request arrives to the storage server and there is not any outstanding request for the same object. The request is stored in the server T object time units after the initiation of the timer. Any request for this object that arrives during these time will be batched.

Batching by timeout has less expected latency than Batching by size [40]. The disadvantage for Batching by size is that it has delays for low and arrival rates, that is why Batching by time out is chosen, because the latency here will be the size of the timer.

**V.2.1 Batching by Timeout:**

The process that Batching follows is the following: Service requests for the same video arriving within a short time duration are bunched together and served in a batch by a single data stream. Since requests are general made for different videos, a server can have a number of pending batches at any one time. Because of that a scheduler will select one according to the policy that it follows. In the next lines we will present the different policies that are used by Batching.

Batching follows different static schemes as we mentioned before: First-Come-First-Serve, Maximum -Queue-Length-First and Maximum-Factored-Queued-Length -First. This schemes or policies are also called Scheduled Multicast because the server selects the next batch to multicast according to some dynamic scheduling policy.

1. **The First-Come-First-Serve** technique what it does is that as soon as some server bandwidth becomes free the batch holding the oldest request with the longest waiting time is served next. Because of the multicast all customer requests for the same movie will be served by the same stream. That is called merging. To select the correct merging policy we will decide if the respective Batching protocol is worth or not. This policy serves the video requests independent of the identity of the movie, i.e. does not matter if it is a popular
video or not. First comes first serve protocol has a deviation that derives in another protocol. The called First-Comes-First-Serve-\(n\) protocol. Both protocols are so similar except that the second protocol reserves a fraction of the server capacity and also preallocates it for batching requests for the \(n\) most popular videos. For all videos with dedicated streams a new stream is started every \(t\) minutes. This amount of minutes will be the batching interval.

There are some advantages in this protocol that in its ancestor are missing. The first one is that with this \(t\) time interval we are fixing the maximum waiting time for the request. This property will favour popular videos. The second advantage is that this policy guarantees a high acceptance probability with a relative small server capacity because the less requested movies do not interference with services for the most popular videos. The side effect is that big number of \(n\) will be inefficient since a lot of reserved capacity for the movies that are not so hot will be underutilized.

To sum up with this scheme we have to note that the rest of the video, the cold videos, will be served following the original First-Comes-First-Serve policy. Using the First-Come-First-Serve-\(n\) scheme that would be First-Come-First-Serve-0 scheme. On the other hand and to finish we should say that if there is not any request for a hot movie we can use its stream to serve a cold movie.

2. The second policy, the namely Maximum -Queue-Length-First what it does is that the batch with the most number of pending requests is chosen to receive the service. What it simply does is the following: Requests for each movie join a separate queue and the movie with the maximum queue length is selected for multicast. But the side effect of this protocol is that popular videos will be favoured respect to non popular ones because they will have the longest queue since they have more requests.

3. On the other hand, Maximum-Factored-Queue-Length-First selects the video \(i\) with the longest queue weighted by a factor \(f_i^{-\frac{1}{2}}\) to be delivered when a server channel becomes free. \(F_i\) denotes the access frequency or the popularity
of video i. In this way not only popular videos will be served, i.e. it does not favour popular videos.

In the following figure we will depict these three policies mentioned above:

Figure 14: Representation of the three batch policies.
V.2 **DYNAMIC MULTICAST APPROACH**

The dynamic multicast approach extends the static multicast approach allowing late coming requests to join a batch currently being served extending the multicast tree to include the new arriving client. In this group will find two different protocols: Patching and Adaptive Piggybacking. We will see like in Batching how the Adaptive Piggybacking protocol has also different policies. This approach offers True Video-On-Demand and provides a high throughput. This is achieved by letting late arriving requests for the same video to be serviced by dynamically expanding the already constructed multicast tree.

V.2.1 **PATCHING:**

Patching was invented in 1998 by Kien A.Hua, Ying Cai and Simon Sheu. With this protocol we will show how can be improved the main problem that static multicast has. As we mentioned on the above paragraph static multicast cannot serve True Video-On-Demand. In Batching we made a client to wait in a queue for a while until his/her request can be broadcasted but with this new alternative we will see how this delay is not necessary and thus True Video-On-Demand is possible reducing cost at the same time by using multicast. We should remind that the problem of True Video-on-Demand is its high cost because before of the use of multicast it was achieved by using dedicated data flow for each request. What Patching does is to let new clients to join an existing multicast. With this we avoid the delay and so, we achieve to have a True Video-On-Demand. The problem of Batching is that it cannot support a True VoD service, i.e. it has a client wait-time. So, Patching deals with this problem. As we can remember with Batching protocol there are two problems. The first one is that users making the early requests are likely to renege if they are kept waiting too long. The second is that if we keep the waiting times short then the benefit of multicast diminishes. To solve these two problems what we have to achieve on the one hand is that waits must be very short for all requests independent of their arriving order and on the other hand each multicast must still be able to serve a large number of users.
With Patching an existing multicast can expand dynamically to serve new clients. Allowing new clients to join an existing multicast improves the efficiency of the multicast. Furthermore, since all requests can be served immediately, the clients experience no service delay.

Within this protocol clients arriving close in time form a session. The server begins to multicast the entire stream at the client playback rate upon the arrival of the first client. The following clients recove the stream from the multicast channel and obtain the missed initial portion, called \textit{patch}, from the server over a unicast channel. In few words a new service request can exploit an existing multicast by buffering the future stream from the multicast while playing the new start-up flow from the start. Once the new flow has been played back to the skew point, the catch-up flow can be terminated and the original multicast can be shared.

Another inconvenience that Batching shows is that each channel multicast the video in its whole so the channel is busy until all the video is played. Hence, each channel can serve less requests and as a consequence the system gets more expensive.

In the following page we will explain step by step all the process that this protocols follows. Patching also splits the physical channel into logical channels. Each channel will broadcast the video at its playback rate. All the request will be allocated in a waiting queue and it will be patched when a channel becomes free. On the other hand, each channel has a client list. This list is formed by the IDs of all clients that make a request. When a channel becomes free the server will look into the waiting queue and depending on the scheduling policy that the Patching protocol has decided to use will change the performance of it.

The advantage of Patching is that unlike the protocols presented before it tries to serve as much request as it can each channel per time unit. The key to reduce this is reducing the time necessary to serve each batch.

We will see a clear example that will illustrate all the theory explained above. We will also see that in patching the client has to download data from two channels at the same time.

We will consider two channels $C_1$ and $C_2$. $C_1$ multicasts the video $v$ during three minutes. After that the channel $C_2$ that was busy becomes free and will batched a new group, $B_2$, but it requests the same video $v$. Imagine that we have two hours video as we are having throughout
thesis. The clients of the new batch can buffer the video part broadcasted in batch B₁ and continue playing the rest from B₂. After three minutes when the catch-up flow has been played back to the skew point, C₂ can be released and the multicast of C₁ can be shared by both batches.

Although the patching data can be consumed as soon as they arrive, the shared data on channel C₁ in this case must be temporally buffered to the local disk.

In the figure below, we will see how the process looks like:

As we can observe in figure 14 we have two kinds of channels: the regular channel and the patching channel. The regular channel is the channel that broadcasts the video in the regular way and the patching channel is the channel in which missing parts are broadcasted. In the figure we see how each client has two loaders, the regular one and the patching loader.
V.2.2 **ADAPTIVE PIGGYBACKING:**

Adaptive Piggybacking appeared in 1996 by Leana Golubchik, John C.S. Lui and Richard R. Muntz. It is the last version of protocols that appeared around user centered. In protocols explained before we showed how batching groups requests for the same video during a time and then it serve them together doing the client waiting in this way. On the other hand, we saw that in Patching also batches are used but in this case if a client arrives late to do a request can continue with the one that is already started and it can recover the missed piece from the patching channel.

Adaptive Piggybacking works in a different way respect to the previous two protocols. It alters the display rates (slowing down or up) of requests in progress for the same object to merge their respective I/O streams into a single stream, which can serve the entire group. It is another protocol used in Video-On-Demand that looks like Batching but the difference between them is that in piggybacking the grouping is done dynamically, while the displays are in progress.

While the I/O bandwidth is saved using this protocol. It is not as high as in Batching because some time has to pass until the streams merge. Another characteristic is that does not use buffers. There are different policies within piggybacking: These policies differ in the way of merging. They are the followings: baseline policy, odd-even reduction policy, simple merging policy, greedy policy and limited merging.

Before we explain them we will explain some basic concepts that are useful to really understand the basic process and the following policies:

\[ S_k = \text{display speed in frames/seconds of display stream k if there is not merge, where } k \in i,j \]

\[ S_k = \text{adjusted display speed of display stream k if there will be merging} \]

\[ S'_k = \text{display speed of display k after merging} \]

\[ p_m = \text{total number of frames in a video object} \]

\[ p_k = \text{current position in object’s display of I/O stream k} \]

\[ p_m = \text{position in frames in an object’s display where I/O streams i and j merge.} \]
$C_k = \text{I/O bandwidth (bits/sec) of I/O stream corresponding to display stream } k \text{ with a display speed of } S_k$

$C_k = \text{I/O bandwidth (bits/sec) of I/O stream corresponding to display stream } k \text{ with a display speed of } S_k^*$

$d = \text{distance in frames between I/O streams } i \text{ and } j, \text{ which is equal to } p_j - p_i$

$d_m = \text{distance in frames between the merge point and the current position of } j, \text{ which is equal to } p_m - p_j$

Before concentrating in the different policies that this protocol has we will explain the general process:

Each display stream, for example the stream $i$, is identified by its position in the object’s display $p$, and is moving at a particular display speed, $S_i$. In order to merge I/O streams $i$ and $j$, firstly, we have to insure that $S_i > S_j$. On the other hand, we have to define the distance and cost constraint which can be used in any Adaptive Piggybacking policy to see if it is possible to merge streams $i$ and $j$. Once that we define these parameters we apply the merging policy and calculate its performance based in the distance. The general parameters are calculated as follows: first of all, we will calculate the cost constraint; it assures that the total bandwidth with merging is less than the bandwidth without it:

$$\frac{dC_i}{S_i} + \frac{d_mC_i}{S_i} + \frac{d_mC_j}{S_j} + \frac{(p_M - d - d_m - p_i)C_i^*}{S_j} \leq \frac{(p_M - p_i)C_i^*}{S_i} + \frac{(p_M - p_i - d)C_j^*}{S_j}$$

(44)

The next equation that we have to define is for the distance constraint. We know that the video is finite and hence,

$$p_i + d + d_m \leq p_M$$

(45)

With equation 45 we defined the distance constraint for a video. The last constraint to define is the merge time constraint that says the following:

$$\frac{d + d_m}{S_i} = \frac{d_m}{S_j}$$

(46)
Having these three equations we can deduce the following distances: let \( d_1 \) be the maximum \( d \) such that the first condition of the bandwidth is satisfied. First, we will get \( d_m \) from equation 46 and then using this new equation we will substitute in the 44th equation. In this way we arrive to have \( d_1 \):

\[
d_1 = \left[ \frac{\left( p_M - p_j \right) C_i^* + \left( p_M - p_i \right) C_j^* - \left( p_M - p_i \right) C_i^0}{S_i} \right] \left[ \left( \frac{C_i}{S_i} - \frac{C_j}{S_j} \right) + \left( \frac{S_j}{S_i - S_j} \right) \left( \frac{C_i}{S_i} + \frac{C_j}{S_j} - \frac{C_i^*}{S_i^*} \right) \right]
\]

Now that we have \( d_1 \) we need \( d_2 \). Let \( d_2 \) be the maximum \( d \) such that the distance constraint in equation 45 is satisfied. Again, \( d_2 \) can be obtained substituting the expression for \( d_m \) into equation 45 and solving for equality:

\[
d_2 = \frac{\left( p_m - p_i \right) (S_i - S_j)}{S_i}
\]

Now that we have the two distances we assume that \( d^* \) is the maximum distance between two I/O streams that when we merge them at \( d_m \) we will get the reduction of the bandwidth. So with the two distances that we got before we deduce that:

\[
d^* = \min(d_1, d_2)
\]

This distance, \( d^* \), will be the key for all calculations and in the policies is denote as a catch up window.

The key in Adaptive Piggybacking is that as sooner happens the merging more resources will be conserved and used by the storage system to service other requests. To explain the different merging policies that adaptive policy uses we have to define some parameter first. We will have just three cases or rates. The minimum or slowest rate, namely, \( S_{\text{min}} \), the normal rate \( S_n \) and the fastest rate, namely, \( S_{\text{max}} \). The cost rates of I/O demands will be \( C_{\text{min}}, C_n \) and \( C_{\text{max}} \).
Below we have the different policies: Baseline Policy, Odd-even reduction Policy, Simply merging policy and the Greedy Policy:

- **Baseline policy:** When the request arrives we do nothing regarding to the display rate. All requests will have the normal display speed and there is not any merging in the system.

- **Odd-even reduction policy:** The goal is to pair up consecutive arrivals to reduce the I/O demand. It must be defined a catch up window. Its size will be the beginning of the object’s display.

![New arrival](image)

The figure above shows one example of how this policy works. When d arrives to the system as we can see c is in the catch up window, $W_{OE}(0)$. As it is shown in the graph, c is at a speed of $S_{min}$ so, d will be $S_{max}$. Likewise, when b arrived also a was in the catch up window so, its display rate is set to $S_{max}$. So a and b merge in the same stream, and the same happens with c and d.

- **Simple merging policy:** Here a catch up window is defined as in the policy above. But it will be defined another window that refers to the latest possible position where two streams can merge and it is called the maximum merging window. The basic idea in this policy is to do merging groups, so, if stream i is the first in the group all the streams that arrive in the catch up window time will merge with it.
If i arrives to the system and finds j \( W_{sm}(0) \) frames ahead of it, then i and j can still merge.

- **Greedy Policy:** What it does is to merge as many times as possible the I/O requests while the object is in display. Here two catch up windows are used. The first is the catch up window that we have in the other policies and the second one is related to the position in the object’s display. This is used as an indication of opportunity for further merging.

We can observe with the graph above how works the Greedy policy. Upon arrival of a request for the object, the speed is performed as in the odd-even reduction policy. If when we cross the catch-up window the stream determines that it has not been paired up for merging, then it checks \( W_{G}(W_{G}(0)) \) to see if there is a possibility to merge with some stream in front.

When merging occurs at position \( P_i \), a new catch-up window \( W_{G}(P_i) \) is computed. If there is not any I/O request within this window, the request’s speed is set to \( S_n \) but on
the other hand, if there are requests within the catch-up window $W_G(P_i)$ and the I/O request that is in front has a display speed of $S_n$, then the request’s speed will be set to $S_{\text{min}}$ and the speed request that is in position $P_i$ is set to $S_{\text{max}}$.

Until this point we have explained how works generally the Adaptive Piggybacking method and now we will explain the different merging policies that exist within this protocol.

- **Limited merging:** Here we will suppose that merging cannot occur any time. Sometimes a replication of data will be necessary in order to perform the display rate alteration. So, for that another parameter is created. This parameter will have the amount of additional disk space that would be necessary to store replicated data. When we apply piggybacking there is a tradeoff between the quantity of the additional storage needed to replicate data and the reduction of the I/O demand that can be. So, to evaluate it we need this new parameter mentioned above. It is the maximum merging point. The merging will be allowed only if it can happen within a specified amount of time or within a certain distance from the beginning of an object’s display; the distance is $P_i + d + d_m \leq p_m^{\text{max}}$. The merge must occur before $p_m$. This last parameter $p_m$ is the position in an object’s display where I/O streams $i$ and $j$ merge.
CHAPTER VI:

VI.1 GENERAL CONCLUSION & COMPARISON

Through all thesis we have been analysing all the notable protocols that nowadays are used in Video-on-Demand. We grouped them using different criterias such as the way in which they split videos into segments, the bandwidth rate that they assign to each logical channel, the frequency in which they broadcast and if they are data-centered or user centered. All this characteristic permitted us to be able to see which protocol is the most suitable in each case.

We remarked at the beginning of the thesis that the main issues in Video-On-Demand Broadcasting are the required bandwidth, the storage capacity needed and the access time that for the client. We have seen that the simple Conventional Broadcasting is not suitable at all because although its simplicity the bandwidth that requires and the waiting time for the client are so high as well as the storage capacity that needs. Based on that protocol that broadcasts all video one after the other through the physical channel we have continued analysing new protocols refining each aspect that was thought not to be correct at all or thinking that could be improved letting us having a better result.

We have started with protocols that are data centered. This protocols as we can remember dedicate the bandwidth to the objects. We have analysed a group of protocols that split the
videos into segments of different sizes and in an increasing way. We have seen how this group of protocols, being the Pyramid Broadcasting protocol the pioneer, has as a main characteristic the notably reduction on the access time for the client but we could also see that the buffering that they need is still so big.

Although Pyramid Broadcasting protocols were a big advance and improvement in Video-On-Demand suddenly appeared new protocols that remove them. The new group of protocols is the Harmonic Broadcasting protocol. In this group we’ve seen that segments are divided into equal sizes but at different bandwidth rates and in this way we could observe that the improvement in the bandwidth is quite big but as we observe later on in these protocol is the problem on delivering the data on time. We then see that there are different kinds of Harmonic protocols. These new Harmonic protocols try to fix the problem that the original Harmonic Broadcasting protocol has.

After analysing these protocols and see their advantages and disadvantages we concluded saying that the best option is to have a hybrid that takes the advantages of both kind of protocols. This new protocol is the Pagoda Broadcasting protocol. This protocol divides the video into equal segments of equal bandwidth but broadcasted into different frequencies thus the improvements are quite big.

After this analysis of Broadcasting protocols based in data centered we have introduced into user centered Broadcasting protocols. These protocols are protocols that dedicate a channel to users instead of objects. We saw how in this case the data centered ones are more scalable because a number of users can be accommodated in the same logical channel. Moreover, in data centered approach the bandwidth requirement is not so big because is proportional to number of videos.

On the other hand, a user centered algorithm dedicates channels in the side of individual users. Moreover, user centered gives better control to the user over the playback of the video but the bandwidth required by the network increases proportionately with the number of users.

The mode of service that user–centered uses, is the client-server approach. In this group we have seen the three models: Batching, Patching and Adaptive Piggybacking. We could no explore them as much as the other Broadcasting protocols. But we showed how they work.
These protocols are able to offer True Video-On-Demand and thus they are so attractive. The problem is that as they are user centered protocols when the number of users increase the network will be exhausted. At the beginning of the thesis we said that these protocols are based in the assumption that the videos that are broadcasted are popular videos and they are transmitted during a time interval. Hence, Batching, Patching and Adaptive Piggybacking will not give any problem as they will be used for a short time and during that time there will be a dedicated bandwidth for the users.

There is not the best solution between protocols that are data centered or protocols that are user centered. In each case and depending on the situation on or the other will be better. After analysing these two groups we can conclude saying that the best option would be to have something that has the positive properties of both, i.e. to have a mix of data and user centered. That is the called hybrid approach. In this case to get the best performance the ideal would be to have a user centered Broadcasting based in P2P. With P2P architecture we will need less bandwidth requirement and also less access time for the client. The main server will broadcast the video to less users and this users will be the responsible to broadcast it again to another users and so on until all users get their video.

A future work thus, would be to analyse the True Video-On-Demand protocols based on P2P.

To sum up we must say that as we have been through the thesis we have been more concentrated in the bandwidth or the access time of the client than in the storage requirement for the client. The reason is that nowadays the storage requirements on the client side are not more a problem. Each client has enough disk and moreover each user has a big storage capacity for not so much money.
CHAPTER VII

APPENDIX

Deployed Systems: YOUTUBE AND CNN PIPELINE

Until now we have been analysing the protocols that nowadays are used in Video-On-Demand. In this section we have a brief explanation about two of the most popular systems developed on Video-on-Demand: Youtube and CNN Pipeline.

1. YOUTUBE:

Youtube is a video sharing web site. Each user can download or upload any video that lasts less than 10 minutes. It was founded in February 2005 but until December 2005 was not launched. After a year of its foundation it was purchased by Google Inc.

At the beginning it served 3 million videos and 800 video uploads per day, transferring 16 terabytes each day. In July 2006 the company revealed that more than 100 million videos were being watched everyday and 2.5 billion video views were watched in June 2006. In July of that year there were more than 65,000 uploads per day and in August, The Wall Street Journal published an article revealing that YouTube was hosting about 6.1 million
videos (requiring about 45 terabytes of storage space), and had about 500,000 user accounts.

In order to watch videos in this site the user must have the following requirements:

1. Macromedia Flash Player 7.0+ plug-in
2. Windows 200 or higher with latest updates installed
3. Mac OS X 10.3 or higher
4. Firefox 1.1+, Internet Explorer 5.0 + or Safari 1.0 +
5. Broadband connection with 500+ Kbps

Youtube limited videos’ size and quality. The size is limited to 320 by 240 pixel dimension and the quality is limited to a bitrate of 314kbit/s with a frame rate dependent on the uploaded video. Youtube limits the playback size and quality by re-encoding the user’s uploaded video at the time of upload. Its playback technology is based in macromedia’s flash player 7 and uses the sorenson Spark H.263 video codec. It requires a plug-in but adobe considers that the Flash7 plug-in is in the 90% of the computers. Youtube converts videos into .FLV(Adobe Flash Video) format after uploading. It accepts to upload videos in .WMV, .AVI, .MOV, MPEG and MP4. Users can view videos in windowed mode or full screen mode and it is possible to switch modes during playback without reloading it due to the full-screen function of Adobe Flash Player 9. Youtube files contain an MP3 audio stream. By default, it is monoencoding with 65 kbit/s rate at 22050 Hz. However, it is possible to get a stereo audio track if the movie file is manually converted to FLV format using a program such as ffmpeg for Linux, ffmpegX for Macintosh or the commercial Riva FLV Encoder for Windows.

2. **CNN PIPELINE:**

CNN pipeline is a news service site. It provides live and on demand videos to the subscribers via broadband Internet connections. It is part of the CNN news group service.

At the beginning it was subscription based and without ads. This system is available since December 5, 2005. Bandwidth, storage and streaming servers are provided by AOL (owned by TimeWarner). AOL means American Online. It is an American global media and Internet services company operated by TimeWarner. It franchises its services
to companies that are in several countries around the world or have set up international versions of its services.

Each of the feeds broadcast in the 16:9 aspect ratio. However broadcast’s resolution prohibited it from being considered a high definition channel so, in June 27, 2007 CNN discontinued the CNN Pipeline service to be a free ad supported live-video streaming media. This service started on July 2, 2007.

CNN Pipeline maintained four simultaneous feeds (or pipes) which had both a primary and a secondary use. From 00:00 to 12:00 UTC on weekdays and all day on weekends, the secondary use was shown on the feeds, meanwhile all other times the primary use was displayed.

When this service was still under payment there were 3 levels of subscriptions:
1. one year $24.95
2. one month $2.95
3. one day $0.99

It streams 4 live video feeds. If one does not want to watch live news then can choose from dozens of on demand news reports, spanning everything from politics and business to sports and entertainment. Some of CNN Pipeline’s stories are drawn directly from the cable television channel but many more are exclusive to Pipeline. Besides giving users the freedom to choose which stories to watch and how long to watch them, CNN Pipeline offers also a broadband tv channel.

<table>
<thead>
<tr>
<th>Pipe Number</th>
<th>Primary use</th>
<th>Secondary use</th>
<th>24 hour broadcast</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Main news program</td>
<td>CNN international multicast</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Raw video feed</td>
<td>CNN new source channel 1 simulcast</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Raw video feed</td>
<td>CNN new source channel 2 simulcast</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>CNN Weather</td>
<td>CNN Weather/Raw video feed</td>
<td>Yes</td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td>----------------------------</td>
<td>-----</td>
</tr>
</tbody>
</table>

Table 13: The 4 live video feeds.

The minimum computer specifications required to use it are the following:

For Windows:

1. Microsoft Windows 2000 with WMP9+ or XP with service pack2.
2. Microsoft internet explorer 6/7 or Firefox 1.0/1.5/2.0
3. Flash player 8+
4. 256MB of RAM but it is recommended to have 512 MB
5. Broadband Internet connection or an access to a high speed network
6. Super VGA(800x600) or higher resolution
7. 16 bit sound card
8. Speaker/headphones

For Macintosh:

1. Mac SX 1.3 “Panther” with Flip4Mac 2.1+, Quicktime 7.1.2+
2. MacOSX 1.4 “Tiger” with Flip4Mac 2.1, Quicktime 7.1.2+
3. Safari 2.0 or Firefox 1.0/1.5/2.0
4. 256MB of RAM but it is recommended to have 512MB
5. Broadband Internet connection or an access to a high speed network
6. Speaker/headphones

The bandwidth requirements to view CNN.com video are the next:
CNN.com video requires a high speed Internet connection for optimal user experience. Broadband connections should be at least 500 Kbps. DSL light and wireless connections may experience slower loading if bandwidth is unavailable for me. CNN does not recommend using dial-up or other low bandwidth connections to access video content.
BIBLIOGRAPHY:

2. Kien A.Hua: Broadcast Techniques for Video-On-Demand
12. Vincet E.P.Heinink: Video-on-Demand over the Internet, 2007
16. Cheng Huang, Jin Li, Keith W.Ross: Can Internet Video-on-Demand be Profitable?, 2007


22. Kien A.Hua, Mounir A.Tantaoui, Wallapak Tavanapong: Video Delivery Technologies for Large-Scale deployment of Multimedia Applications, 2004


24. Huadong Ma, Kang G.Shin: Multicast Video-On-Demand Services, 2002


27. Li-Shen Juhn, Li-Ming Tseng: Fast Broadcasting for Hot Video Access, 1997

28. Li-Shen Juhn and Li-Ming Tseng: Harmonic Broadcasting for Video-On-Demand Service, 1997

29. Lars Engebretsen, Madhu Sudan: Harmonic Broadcasting is Bandwidth-Optimal Assuming Constant Bit Rate, 2006


33. Chi-He Chang: A recursive Frequency-Splitting Scheme for Broadcasting Hot Videos in VOD Service, 2000

34. Huadong Ma, Kang G.Shin: Multicast Video-On-Demand Services, 2002


36. Asit Dan, Dinkar Sitaram, Perwerz Shahabuddin: Scheduling Policies for an On-Demand Video Server with Batching, 1994

38. Ying CAi, Kien A.Hua: Sharing Multicast Videos Using Patching Streams, 2004


45. http://www.pcmag.com/article2/0,2817,1896416,00.asp


48. http://www.youtube.com/t/about