UNIVERSIDADE DO ALGARVE

REAL-TIME TRANSMISSION OF SCALABLE VIDEO OVER PEER-TO-PEER NETWORKS

Pedro Miguel Larguito Rodrigues

Dissertação para a obtenção do grau de Mestre em
Engenharia Elétrica e Eletrónica
(Área de Especialização em Tecnologias da Informação e Telecomunicações)

Trabalho efetuado sob a orientação de:
Professor Doutor Jânio Miguel E. Ferreira Monteiro

2012

UNIVERSIDADE DO ALGARVE

REAL-TIME TRANSMISSION OF SCALABLE VIDEO OVER PEER-TO-PEER NETWORKS

Pedro Miguel Larguito Rodrigues

Dissertação para a obtenção do grau de Mestre em
Engenharia Elétrica e Eletrónica
(Área de Especialização em Tecnologias da Informação e Telecomunicações)

Trabalho efetuado sob a orientação de:
Professor Doutor Jânio Miguel E. Ferreira Monteiro

2012

Declaração de Autoria de Trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Prof. Doutor Jânio Miguel Evangelista Ferreira Monteiro pela disponibilidade e empenho com que me orientou neste trabalho. Gostaria de destacar os conhecimentos que me transmitiu, a sua disponibilidade com o tempo que me dispensou e a motivação que sempre me transmitiu desde os momentos iniciais deste trabalho.

Gostaria ainda de agradecer à minha família por todo o apoio e motivação que me deram e por fim dedicar esta Tese à minha esposa Lina Ramos, por todo o apoio, dedicação e por ter estado sempre a meu lado em todas as etapas deste longo percurso.

Resumo

Nos últimos anos temos assistido ao expressivo crescimento na procura de conteúdos de vídeo na Internet. Esse crescimento tem surgido associado ao aumento da diversidade dos terminais com capacidades para receber conteúdos de vídeo e ao aumento na procura de conteúdos em alta definição, colocando novos desafios de heterogeneidade e escalabilidade às redes que servem de suporte à distribuição desses conteúdos.

O problema da escalabilidade tem sido resolvido tradicionalmente nas redes IPTV (Internet Protocol Television) recorrendo ao IP Multicast, suportado em redes e equipamentos administrados por operadores privados e que por isso têm mecanimos de controlo que reduzem os riscos associados ao mesmo. No entanto, na Internet, tais riscos levaram a que o IP Multicast não fosse adotado, o que por sua vez condiciona a distribuição em larga escala de vídeo. Neste sentido, os fornecedores de conteúdos vêm-se por isso obrigados a recorrer a soluções ditas de nível de aplicação ou também denominadas de soluções sobrepostas. Neste âmbito as soluções Peer-to-Peer são hoje extensivamente utilizadas como suporte à troca de ficheiros, o que poderia fazer delas uma possível solução à distribuição ponto-multiponto de vídeo.

Em relação ao problema da heterogeneidade de terminais, a introdução recente de normas de compressão escalável de vídeo permitem ir ao encontro da variabilidade de equipamentos com diferentes definições e capacidades de processamento.

Neste âmbito, a investigação efectuada nesta dissertação pretende combinar as soluções Peer-to-Peer mais importantes, com o vídeo escalável, no sentido de obter um sistema que, suportado na Internet, permita a distribuição ponto-multiponto de conteúdos com requisitos de tempo real para um número elevado de terminais com características heterogéneas.

Palavras chave

Distribuição de Vídeo em Tempo Real, Distribuição Ponto-Multiponto, Peer-to-Peer, Conteúdos H.264, Vídeo Escalável

Abstract

In the last years, we've witnessed a significant growth in the demand of video content on the Internet. This growth has emerged combined with an increasing diversity of equipments capable of receiving video content and an increase in the demand for high definition content, placing new challenges of heterogeneity and scalability to the networks that should support the distribution of such contents.

Traditionally the scalability problem has been solved by IPTV (Internet Protocol Television) networks using IP Multicast, supported by networks and equipments administered by private operators, with mechanisms capable of reducing the risks associated with flows that do not respond to congestion. However, on the Internet, such risks have resulted in the lack of implementation of IP Multicast, which in turn affects large-scale distributions of video. This has caused content providers to use so-called Application layer or overlay solutions. In this context, Peer-to-Peer solutions are currently extensively used for file exchange, making them a possible solution to the point to multipoint distribution of video.

Regarding the problem of terminal heterogeneity, the recent introduction of standards for scalable video compression, allows the distribution of serveral qualities, targeting a range of equipment terminals with different resolutions and processing capabilities.

In this context, the research conducted in this dissertation aims to combine the most common Peer-to-Peer solutions, with scalable video, in order to obtain a system supported on the Internet, that allows a point-multipoint distribution of real-time content for a large number of heterogeneous terminals.

Keywords

Real Time Video Distribution, Point-Multipoint Distribution, Peer-to-Peer, H.264 Content, Scalable Video

Table Of Contents

List of Figures

List of Tables

List of Acronyms

AVC	Advanced Video Coding
CGS	Couarse Grained Scalability
GOP	Group of Pictures
HD	High Definition
IP	Internet Protocol
MGS	Medium Grained Scalability
MPEG	Moving Pictures Experts Group
NAL	Network Abstraction Layer
P2P	Peer-to-Peer
QoS	Quality of Service
RSS	Really Simple Syndication
SNR	Signal-to-Noise Ratio
SVC	Scalable Video Coding
TV	Television
VCL	Video Coding Layer
XML	eXtensible Markup Language

Chapter 1

Introduction

In the last years, we've witnessed a significant growth in the demand for video content on the Internet. This growth has emerged from the increasing diversity of equipments capable of receiving video content combined with an increment in the demand for high definition sequences, placing new challenges of heterogeneity and scalability to network operators supporting the distribution of such contents.

The usage of video services over the Internet has spread exponentially all over the globe in the last years; services like YouTube, live video streaming, online video purchases and rentals and webcam viewing have become the main responsibles for the increment of the global Internet traffic.

This rough usage spread, has captured the attention of television channels and broadcasters that look to the Internet as a viable media for delivering their contents to costumers all over the world.

In order to explore this new media distribution path, several different solutions are currently appearing all over the Internet, with the aim to overcome the fact that traditional video distribution solutions based on IP Multicast are not available in the Internet.

One solution that has been extensively used to overcome the scalability problems of the Internet is based in constructing Peer-to-Peer (P2P) overlays in which several terminals cooperate as clients and/or servers. In fact, P2P traffic is currently one of the largest traffic types on Internet, only overpassed in 2010 by Internet video traffic [1]. Due to this, a solution that combines both, video distribution and P2P overlays has therefore a very high importance.

In terms of P2P, the BitTorrent is currently the most popular P2P system on the Internet. However like their predecessors it is fully oriented for file sharing. This results from the fact that until 2003, the main focus of P2P systems was centered in file sharing over the Internet. When Skype

implemented a Voice over P2P application, it triggered the debate for the usage of P2P in different kind of applications [2].

In terms of video, while some services like [3][4][5] are currently capable of delivering live video over P2P overlays, the problems placed by the heterogeneity of terminals in terms of processing power and definition are still far from being solved. In this field, Scalable Video Coding has recently emerged as the solution that permits the delivery of different image definitions, frame rates and signal-to-noise qualities targeting different terminal capabilities. However, while some studies have proposed the real-time delivery of scalable video over P2P [6], few have addressed the real-time encoding and transmission of Scalable Video based in the BitTorrent mechanism.

1.1. Problem Statement

Besides the big number of different solutions currently appearing over the Internet for live video distribution, all these new software applications suffer from very similar problems, which includes the lack of adaption to bandwidth fluctuations, lack of support for different type of terminals, low quality of the multimedia streams due to the lack of support of Quality of Service (QoS) in the Internet and they are proprietary solutions without a defined standard.

Given the fact that IP Multicast cannot be used in video distributions over the Internet, combined with the problems caused by terminal heterogeneity, a solution is required to support P2P transmission of television channels using scalable video coding. Specifically the focus of this study is to understand how to distribute a scalable video bitstream that is being encoded in real-time, over a BitTorrent overlay.

1.2. Dissertation Overview

The research on this thesis aims to the definition of a point-to-multipoint real-time video distribution system, capable of distributing real-time content over the Internet, overcoming the scalability issues of the traditional systems. In order to guarantee a scalable distribution over the Internet, the system is based on a Peer-to-Peer overlay combined with scalable video technology as a mean to support terminal heterogeinity.

In this chapter, the aim of this research is presented; by describing its scope, naming the main objectives.

In chapter 2, a description of the H.264/SVC (Scalable Video Coding) standard is made, namely the description of the bit stream structure and major encoding features.

In chapter 3, a description of Peer-to-Peer networks is performed, focusing in the different topologies of Peer-to-Peer networks, the resource discovery and indexing methods and the different point of views for the operation of the Peer-to-Peer overlay. Finally, a detailed analysis to the BitTorrent protocol is performed, regarding the different components of the swarm, the different messages exchanged between peers, the different roles executed by a peer and a detailed example of a functional scenario.

Chapter 4 describes the initial steps performed to combine H.264/SVC and BitTorrent, in order to implement a distribution system capable of transmitting H.264/SVC over the Internet based on a Peer-to-Peer BitTorrent overlay. A proof-of-concept test environment was setup and the tests obtained were analyzed. With them several challenges were identified. This chapter also presents and describes the implementation of the software used on these tests.

The chapter 5 describes in detail a BitTorrent simulation framework designed for this research and the tests performed with it, analyzing the obtained results, including the number of layers of quality received by the peers and the different delays verified, according to the challenges identified in the previous chapter.

Finally, chapter 6 concludes this dissertation and some future work plans are described.

Chapter 2

H.264 Scalable Video Coding

The H.264 Moving Pictures Experts Group 4 (MPEG-4) Part 10 [7] or Advanced Video Coding (AVC) is a standard for video encoding/decoding. The development of this standard started in 2003, with the main purpose to suit the growing needs for quality in the video services when transmitting at lower bandwidths [9].

Nowadays, we can find this standard in many applications such as Blu-Ray discs, videos from YouTube, web software such as Adobe Flash Player and Microsoft Silverlight, broadcast services for DVB, direct-broadcast satellite television services, cable television services and real-time videoconferencing [10].

Given this global acceptance, a scalable extension of this standard has also been defined with the aim of supporting the transmission of several layers of quality. H.264 Scalable Video Coding (SVC) is the extension of the codec that allows video devices to send and receive multi-layered video streams composed of a base layer and one or more optional layers that enhance the quality of the base layer in terms of resolution, frame rate and/or quality.

The standardization process of this extension has also started in 2003, but only in 2007, after the analysis by the Moving Picture Experts Group (MPEG) of several proposals, the specification for H.264/SVC was finished [7][10].

The aim of this chapter is to describe the structure of Scalable Video Coding which will enable a better understanding of how it can be used to adapt to different terminals in terms of screen definition and processing power. It starts with the description of H.264/AVC, and particularly its bitstream structure as a mean to understand the structure of the scalable extension and its suiting for video distributions over IP networks. Finally, the structure of scalable extension H.264/SVC is described, as well as, its different scalable dimensions.

2.1. H.264 Advanced Video Coding

The structure of the H.264/AVC standard is mainly divided in two layers, the Video Coding Layer (VCL) and the Network Abstraction Layer (NAL). The Video Coding Layer is responsible for the encoding of the video content, while the Network Abstraction Layer formats the VCL representation and provides information to enable the mapping to different network protocols and/or file formats.

The VCL follows a block-based hybrid video-coding approach, similar to the ones found in previous standards. The basic source-coding algorithm uses a hybrid of inter-picture prediction that exploits the temporal dependencies between images, combined with a transform coding of the prediction residual to exploit the spatial statistical dependencies.

In the VLC, each image is fragmented in several fixed size macroblocks. Macroblocks are in turn grouped in slices. Each slice represents subsets of the picture which can be independently decoded. H.264/AVC considers five different slice types, with the most important ones: I, P and B slices. In intra-coded slices (I slices), all macroblocks are coded without requiring information from other images. On the other hand, predictive (P) and bi-predictive (B) slices require information of a previously encoded slice.

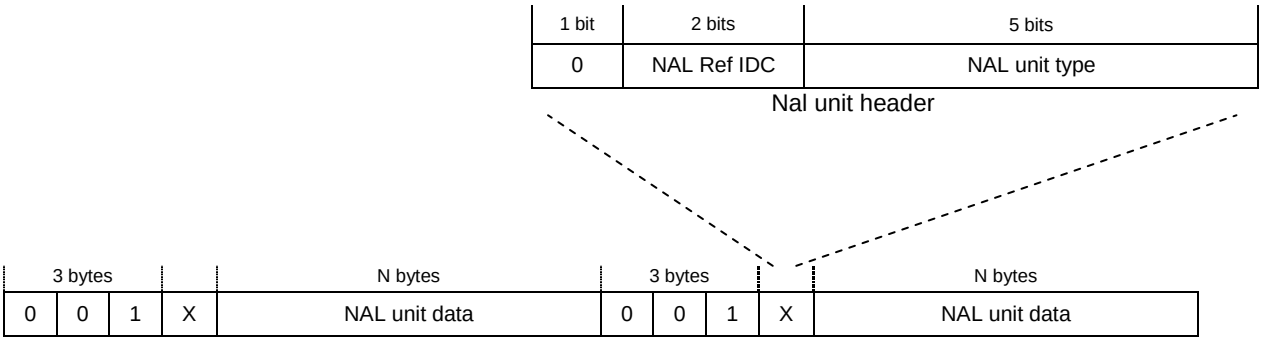

Figure 2.1: NAL unit structure.

After encoding, the NAL formats data and provides header information in accordance to different transport layers or storage media. The H.264/AVC NAL units represent packets of binary data that are delimited by a start code prefix, which is a sequence of 3 bytes with a fixed-value of 0x000001. As shown in Fig. 2.1, the start code is followed by a one-byte header with information about the type of NAL unit and then an integer number of data bytes corresponding to the data structure of the NAL unit. Finally, one or more trailing bytes with the value 0x00 finish the packet. The start code prefix can be preceded with several leading zero bytes before the 0x000001 sequence.

2.2. Scalable Extension of the H.264

H.264/SVC is the scalable extension of H.264/AVC standard enabling the encoding of several levels of quality. The scalability properties of H.264/SVC supports the splitting of the main bit stream into several smaller streams, forming a base layer, fully compatible with H.264/AVC devices, and several enhancement layers. The fragmentation in several layers can be performed by identifying different NAL units in the binary coded video sequence.

Each NAL unit belongs to one of the three basic scalable dimensions, namely: spatial (resolution), temporal (frame rate) and Signal-to-Noise Ratio (SNR) quality. Each NAL unit thus carries a given level of spatial, temporal and SNR quality which might be obtained from three identifiers: dependency_id, temporal_id, and quality_id. They are sometimes referenced by a (D, T, Q) tuple, as described in [11].

2.3. Temporal Scalability

By definition, a bit stream provides temporal scalability when its access units can be partitioned into a temporal base layer and several temporal enhancement layers. Each frame in the temporal layer is marked with an identifier T, which is 0 for the temporal base layer and is increased by 1 from one temporal layer to the next [8].

In H.264/SVC, temporal scalability is achieved by using hierarchical coding structures of predicted frames (B-frames). The frames of the temporal base layer can only be predicted from previous frame of this layer and the frames in the enhancement layer can be bidirectionally predicted by using the two surrounding pictures of a lower temporal layer as references, as shown in Figure 2.2.

A frame from the temporal base layer (I- or P-frames) and all temporal enhancement frames (B-frames) between the base layer frame and the previous base layer frame (I- or P-frames) build a Group of Pictures (GOP). The size of the GOP determines the number of temporal layers that might be achieved.

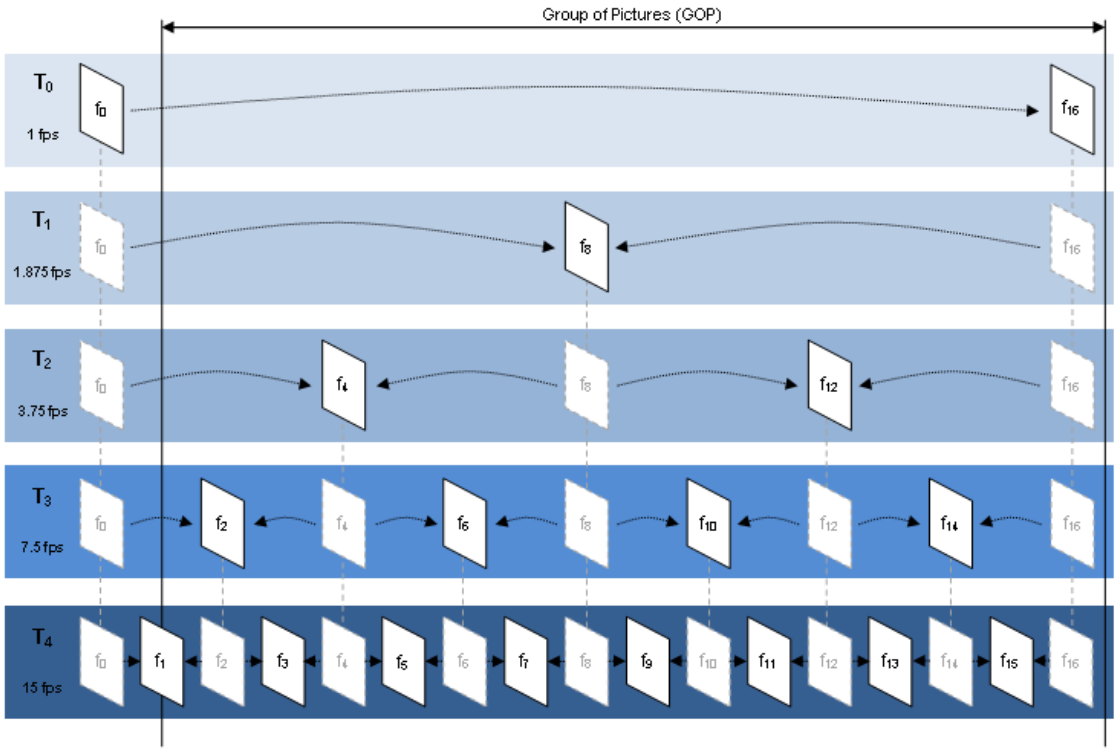

Figure 2.2: Temporal layer encoding structure using hierarchical prediction structure for a GOP with 16 frames.

2.4. Spatial Scalability

Spatial scalability provides support for several display resolutions with arbitrary ratios. In H.264/SVC, spatial scalability follows the conventional approach of multi-layer coding, used in prior video coding technologies like H.262 (MPEG-2) or H.263 [1]. Each spatial layer corresponds to a different spatial resolution and is marked with a dependency identifier D, with value of 0 for the spatial base layer and increased by 1 from one spatial layer to the next.

Similar to the temporal scalability, the spatial scalability also uses motion-compensated and intra-layer prediction techniques for frame prediction within the same spatial layer; however, in order to reduce the spatial redundancy, it also uses inter-layer prediction techniques between different spatial layers, as shown in Figure 2.3.

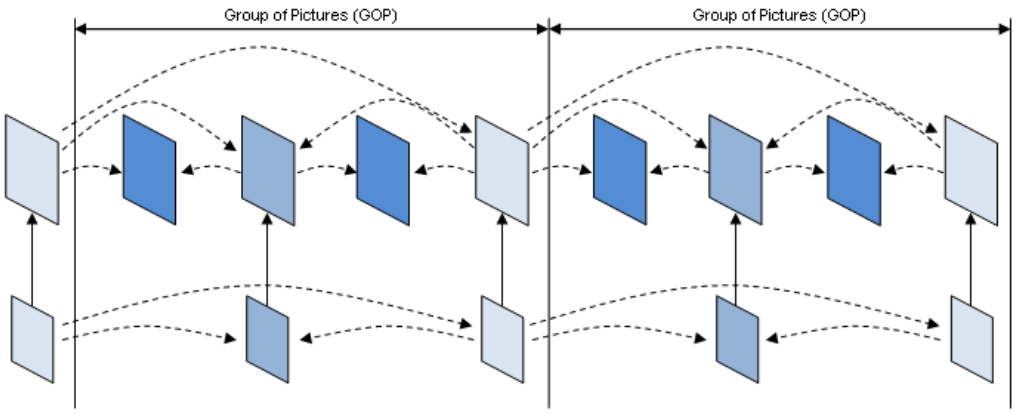

Figure 2.3: Combined temporal and spatial scalable coding considering two different frame rates at lower and higher layers.

2.5. Quality Scalability

Quality scalability describes the support of multiple signal-to-noise ratio quality layers. In H.264/SVC, quality scalability can be considered a special case of spatial scalability using the same picture size for the base and the enhancement layers [1]. Each quality layer corresponds to a different SNR and is marked with a quality identifier Q, which is 0 for the quality base layer and is increased by 1 from one quality enhancement layer to the other.

The similarity between quality and spatial scalabilities, results from the usage of the same inter-layer prediction techniques and a similar motion-compensation technique. This motion-compensation mechanism, named as Medium Grained Scalability (MGS) or Coarse Grained Scalability, is an improved version of the mechanism used for spatial scalability, which on contrary of other options enables a single loop of decoding pictures, normally using the highest available quality for motion estimation and compensation, as shown in Figure 2.4.

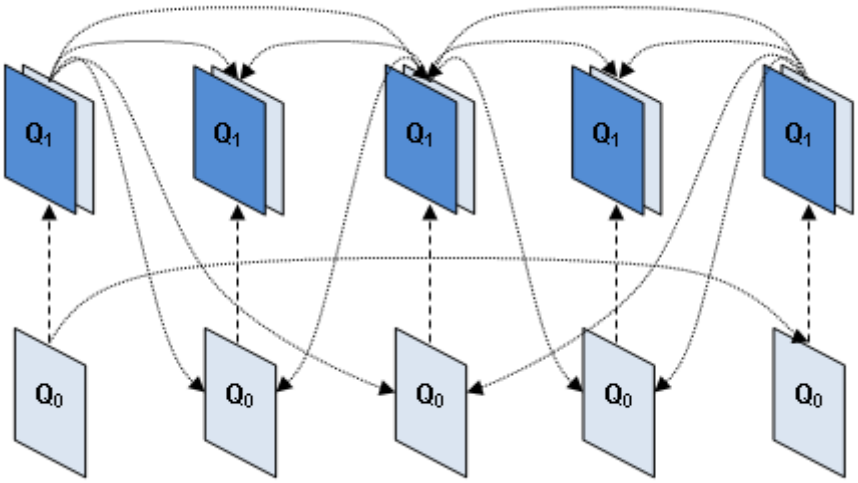

Figure 2.4: Quality Scalability using Medium or Coarse Grained Scalability.

2.6. Combined Scalability

The H.264/SVC coding structure is organized in interdependent layers, combining temporal, spatial and quality scalability layers. In a H.264/SVC video bit stream each NAL unit is identified using the (D, T, Q) tuple, which identifies the level of the spatial (dependency_id), temporal (temporal_id) and quality (quality_id) of the bit stream as shown in Figure 2.5.

This type of structure makes the H.264/SVC bit stream very suitable for the transmission over IP networks, since the fragmentation and reassembly processes of those bit streams into and from the several protocols stacks becomes very simple when compared to other video coding technologies.

However, beyond offering better resilience to errors than H.264/AVC, H.264/SVC it's still very sensitive to packet losses. Several studies have also shown that, besides the interdependency between layers, errors in layers of different scalability dimensions have different levels of impact in the quality of the receptioned video stream [12]. Thus, in order to minimize those impacts, H.264/SVC scalability layers should be properly conveyed in transmission layers.

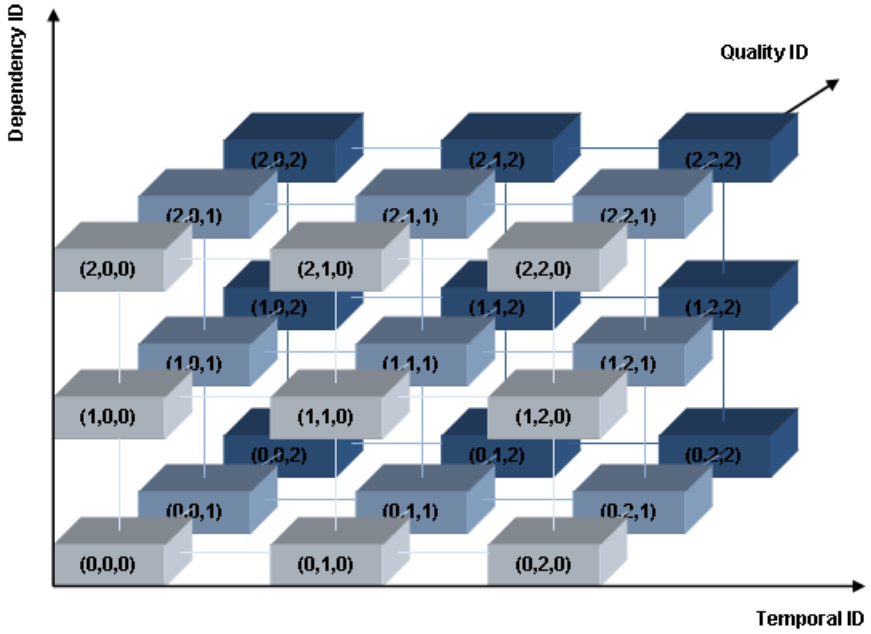

Figure 2.5: Interdependency of a H.264/SVC coded video sequence.

2.7. Summary

The scalability features of H.264/SVC coding structures combine temporal, spatial and quality levels in interdependent layers, in order to provide dynamic encoding and decoding processes of the generated partial bit streams. Each of these generated partial bit streams can be associated with a correspondent transmission layer, forming one for the base layer and several enhancement layers.

When combined, these partial bit streams can reconstruct the original bit stream or similar ones with lower temporal, spatial or quality levels.

The dynamic reconstruction of these bit streams provides significant rate adaptation capabilities to H.264/SVC, enabling functionalities such as the capability to provide video services to environments with a wide terminal heterogeneity and graceful degradation in lossy transmission environments.

The dynamic decoding process of the partial bit streams enables the decoding of multiple temporal, spatial and quality levels, which is a great benefit in terms of the simplicity of the distribution system and in bandwidth savings, when compared to the traditional point-to-multipoint TV distribution systems where different resolutions correspond to the transmission of a different bit stream. These benefits make H.264/SVC very suitable to be used for a point-to-multipoint TV distribution system as further described in this research.

Chapter 3

Peer-to-Peer Systems

The term Peer-to-Peer (P2P) network is commonly used to reference an overlayered system of computers and the paths between them in their underlying physical networks. Each computer can act as a client or server for the other computers in the system, without the need of a central server to exchange content, as typically occurs in client-server based solutions.

The first implementation of a P2P system appeared in 1999, with Napster. Napster was a system fully designed for file sharing which achieved a great success in the Internet. However due to several legal issues regarding the copyright of music contents and their illegal distribution over the Internet, it was closed in 2001. The following P2P solutions, like Gnutella, eDonkey and BitTorrent, continued to gain more and more followers on the Internet. P2P quickly became the largest traffic type on Internet and only nearly 10 years later, in 2010, became overpassed by Internet video traffic as described in [1]. Currently, BitTorrent is the most popular P2P system on the Internet and like their predecessors is fully oriented for content file sharing.

The main purpose of P2P systems had always been the file sharing over the Internet. Only in 2003, with the appearance of Skype, a Voice over P2P application, the debate for the usage of P2P in different kind of applications was raised [2]. More recently, we've witnessed to an emergent growth of P2P based applications in the market, including Voice over P2P and P2P TV.

Given these considerations, the aim of this chapter is to describe Peer-to-Peer systems, supporting a better understanding of how they can be used in real-time video distribution systems over the Internet, as an alternative to the IP Multicast distributions, used in privately administrated IPTV networks. This chapter also includes a detailed description of the BitTorrent protocol as a mean to understand its characteristics and functionalities, since it will be extensively used in this study, for the distribution of real-time video content.

3.1. Peer-to-Peer Topologies

A Peer-to-Peer system is often implemented by an overlay network on top of another network, like a physical network or the Internet itself. The nodes of these overlays, also called as peers, are virtually connected to other nodes within the overlay, forming many logical paths between them. The overlay is responsible for making the P2P system independent of the physical network topology, so the content can be directly delivered over the underlying IP network. The overlay is also the responsible for the indexing and discovery of new peers [2].

Peer-to-Peer overlays can be classified according to the following topologies; pure, centralized and hybrid [12]. In pure or decentralized P2P overlays, peers exchange information between them without requiring any central server to keep track of the location of the file pieces or to verify user credentials. Examples of such P2P overlays are Gnutella and Freenet systems.

On the contrary, in centralized P2P overlays one or more central servers are used and peers contact them before contacting other peers. The usage of these central servers brings some disadvantages to these systems, including the possibility of congestion of those servers and their vulnerability to attacks. Napster and BitTorrent are examples of centralized overlays.

In hybrid P2P overlays there is a hierarchy of nodes, with different nodes having different tasks to perform. Super-peers or supernodes are in those cases used to discover resources on behalf of other peers, and to answer their queries. Systems like Kazaa and eDonkey are examples of hybrid P2P overlays.

3.2. Resource Discovery and Indexing in Peer-to-Peer Overlays

As described in the last section, content transmission and routing between peers of a P2P overlay rely on the underlying networks connections, established and maintained by IP routing protocols.

Different methods can be used to identify and find content within an overlay network.

Peer-to-Peer overlays can be categorized in two different types of overlay graphs; the unstructured overlays and the structured overlays.

In unstructured overlays, content discovery is based on random topologies, which are built based on the peer simple neighboring knowledge and for this, the most commonly used mechanisms are flooding and random walk routing. Many applications rely on these types of random graph

implementations, which have normally shown several scalability problems when the number of peers increases significantly. Gnutella, Freenet and FastTrack are examples of implementations of unstructured P2P systems [12].

In structured overlays, content discovery is based on more accurate routing mechanisms maintained by the peers. A key is assigned to each object identifier, derived from information about its title, author and format. The system then routes each of these keys along the overlay network and according to the peer address identifiers, stores it in a special node. These special nodes are then responsible for answering queries for similar content searches. These systems have shown to be more scalable than unstructured systems, but in contrast, are more prone to problems when peers join and leave the overlay. Chord, Tapestry and Kademlia are examples of implementations of structured P2P systems [13].

3.3. Peer-to-Peer Operation

Peer-to-Peer systems are very useful for the distribution of the same content to a large number of users at the same time over the Internet, since the content is stored and shared cooperatively between a large number of users.

In general, the operation of a P2P system can be described in two points of view: the user point of view and the overlay network point of view [2].

In the user point of view, the user downloads a P2P software client from a website on the Internet and installs it on his computer. After launching the P2P application and since the computer is already connected to the Internet, it attempts to connect to certain hosts on the Internet that are configured in the software for bootstrapping purposes. It uses these connections to find other peers to connect to so that it can join the overlay. Initially it starts with a few connections to other peers, but gradually as it exchanges information with peers, the number of connections increases.

Even if the user hasn’t selected any files to download, or selected local files for sharing, the P2P application is most likely using the computer and its network connection in support of other peers, as for instance responding to search requests.

Later, when the user wants to start downloading files from the P2P overlay or shares its own files to the other users, the P2P application will send messages to the peers to which it is connected with, in order to search for content, or to send information about the files the peer is sharing.

The point of view of the overlay starts with the formation of the overlay itself, in the same instant that a group of computers becomes interested in forming a P2P network. At this point, a bootstrap mechanism is used to connect peers between them, since they may be distributed on the Internet and will have to find some way to discover each other and to form the initial overlay. Several options exist to be used as a bootstrap mechanism, like: using a well known server to register the group of peers; a multicast group address for peers to join; or using local broadcasts to collect nearby peers, followed by progressively merging these peer sets into larger sets until the overlay is formed.

The overlay is a logical layer for delivering messages and content between peers, where peers may join or leave the overlay at any time, originating fluctuations in the overlay's shape.

Since peers typically store content needed by other peers, they also need to search the overlay for information of their interest. The search mechanisms used in P2P are distributed across the entire overlay, using for instance a distributed search system where each peer can route a search query to the correct peer. An efficient search query routing is an important aspect of a P2P system design.

The overlay routing of messages and search mechanisms are the major blocks to build different types of P2P applications.

3.4. The BitTorrent Protocol

The BitTorrent protocol [14] is the most popular P2P system currently used on the Internet. It was designed to distribute large files, fragmented in small pieces, using a mutual distribution method between a group of peers, called a swarm.

BitTorrent follows a centralized P2P topology, using one or more central servers to keep track of the peers in the swarm. These central servers are named trackers. A tracker is a modified web server, which contains updated information about all the peers participating in the swarm. Peers exchange information about their location and availability with the tracker periodically, normally every 30 seconds; peers also use trackers as bootstrap mechanism when joining the swarm.

BitTorrent uses an unstructured overlay graph with a combination of well known strategies and the information of the neighboring peers to discover and download content.

Firstly, a peer indicates if it is interested in receiving exchanges from a neighbor peer in the swarm. If it becomes interested, a rarest-first prioritization criteria is used to request file pieces for download from the neighboring peers. By selecting the rarest pieces first for download, the peer

contributes to increase the existences of a file piece in the swarm and with it increases the probability of other peers in the swarm waiting to download the same file piece.

Somewhere in the swarm, the uploading peer applies a tit-for-tat strategy for choosing to which peers to send the requested file pieces. This tit-for-tat strategy, consists in a fair exchange mechanism with the neighboring peers, where each peer should be reciprocate by supplying pieces to peers that also provide downloads to it, favoring the peers with higher capacity of transferring information, based on previous data transfer rates with each neighbor peer.

The purpose of each peer is to maximize its piece download rate in a reciprocal manner. The implemented strategies can achieve fairness and provide incentives for mutual exchanges, but to avoid overloading and to achieve a good TCP performance, each peer limits the number of simultaneous active connections, typically to four different peers [2]. Then a choke/unchoke mechanism is put into place and the active connections are placed in the unchoked state, while the other neighbor connections are placed in the choked state. To avoid repetitive changes of choked state, called as fibrillation, this process is limited to occur in intervals of 10 seconds.

Additionally, each peer tries to connect to other peers periodically to see if their download rate is better than the current ones in its active group of peers. This optimistic unchoking mechanism selects an interested peer randomly every 30 seconds.

When data is being transferred, peers use a technique called pipelining to get good transport performance. This technique consists in sending several requests for file pieces messages at once without waiting for a response from the remote peer, as shown in Figure 3.1.

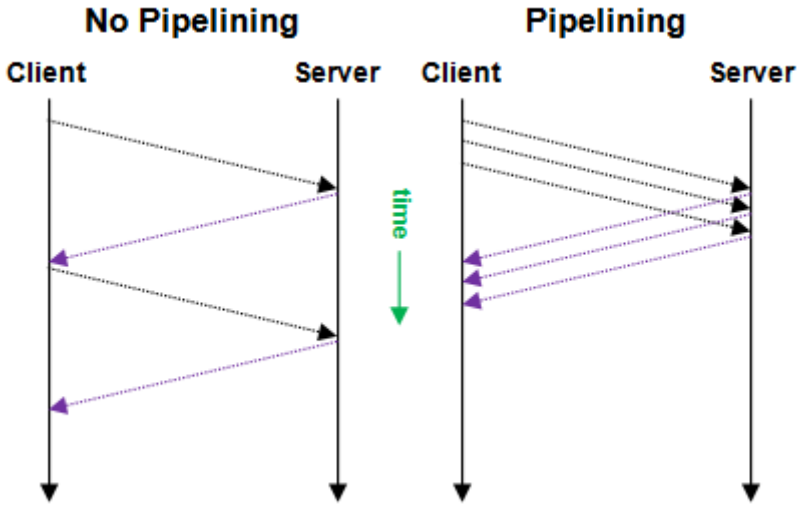

Figure 3.1: Example of a peer downloading files from a BitTorrent swarm.

Active peers with a complete copy of the content are called seeds and peers still downloading the content are called leechers [14].

Finally, besides requiring a BitTorrent software installed on a computer, a very important component is required to download a file using BitTorrent protocol: the torrent file. The torrent file contains very important information required for downloading content files, which includes the name and size of the content files, the URL of the trackers of the swarm and the fixed piece size in which the content files will be chopped into.

3.5. Example of Data Exchange using BitTorrent

Figure 3.2 a), b), c) and d), shows a step-by-step example of a new peer joining a fully functional BitTorrent swarm and then downloading the content files in a torrent file.

Assuming that BitTorrent software was previously installed on the peer of this analysis (highlighted in green); the first step to start downloading files with BitTorrent is to obtain a torrent file. The torrent files are normally downloaded from a web page somewhere on the Internet, represented in Figure 3.2 a) by message 1. After the BitTorrent application gains access and reads the content of the torrent file, it sends a message (2.) to the respective tracker, announcing the interest of that peer in joining the swarm. The tracker processes the received message, updating its internal database with the information of the new peer and sends back a response (represented by message 3.) with a list of other active peers that are participating in the swarm. Then, the peer tries to contact every other peer in the list, in order to perform a handshake, i.e., to exchange information about their location, the interest in exchanging file pieces and the file pieces availability.

Peers continuously exchange information about the availability of pieces between them, as represented by messages 5. in Figure 3.2 b).

Using that information, peers request to their neighbours each of the missing pieces using a rarest-first prioritization criteria and afterwards waits for their reception (as shown in Figure 3.2 b)).

Soon, one or more neighboring peers will unchoke their connections with the peer and begin sending the requested pieces (Figure 3.2 c)). As the requested file pieces arrive, the peer will send messages to all the neighboring peers to inform them of the availability of the new piece files (represented by messages 8. in Figure 3.2 c)).

The steps described in Figure 3.2 b) and Figure 3.2 c) repeat until all the pieces of that file are downloaded.

Meanwhile, the peer can also receive requests for missing pieces from the neighboring peers and send them to the neighboring peers, this can be observed in Figure 3.2 d) (in messages 9. and 10.), where the peer reciprocally starts seeding pieces to a neighbor peer.

Peers and trackers exchange messages in a periodical time basis, normally 30 seconds, in order to update the tracker's information about the active peers and to give information about new peers joining the swarm.

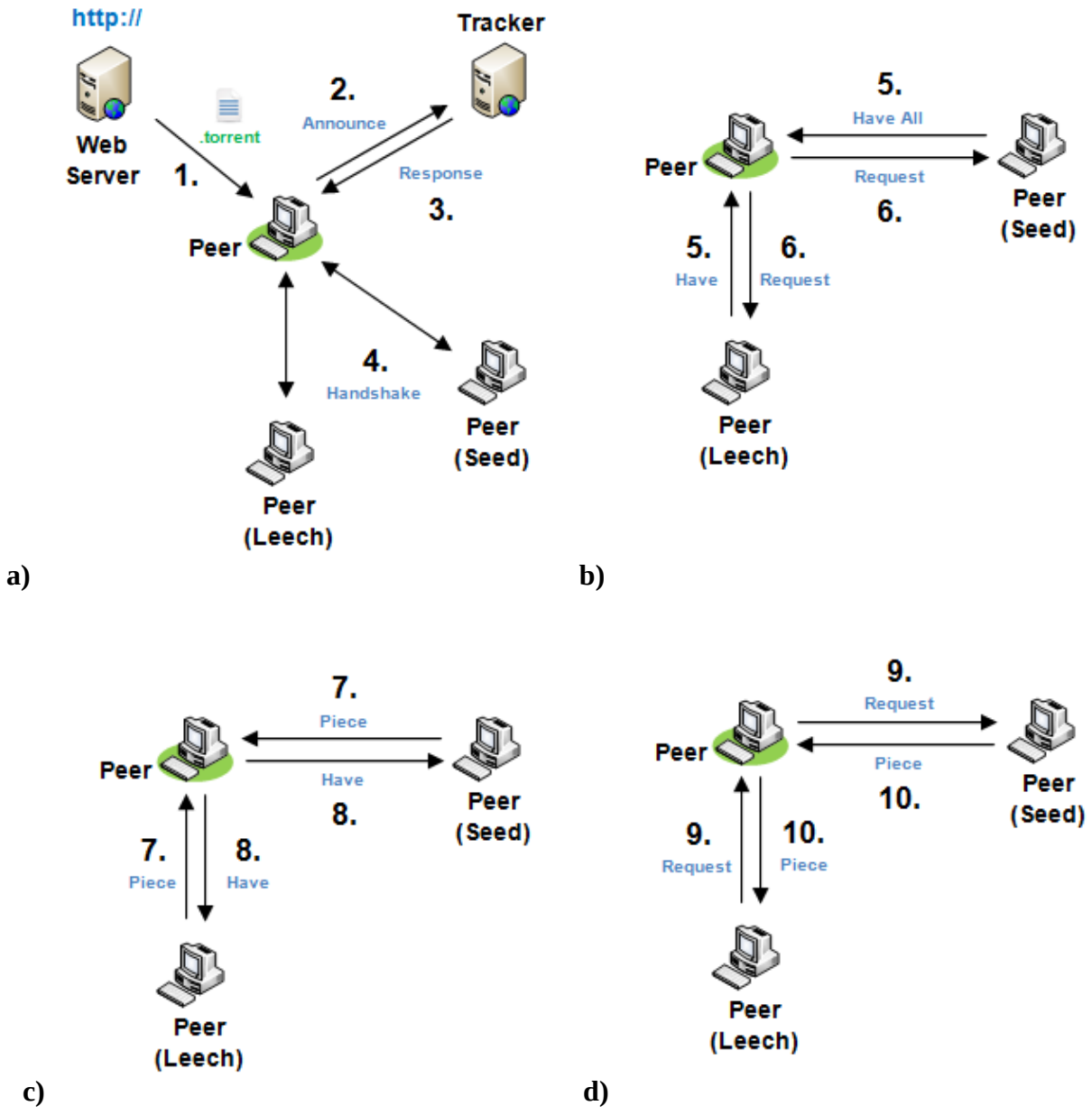

Figure 3.2: Example of a peer downloading files from a BitTorrent swarm.

3.6. Summary

The usage of Peer-to-Peer systems and applications over the Internet has grown exponentionally in the last years, mainly due to the sudden growth of an enormous variety of applications using these network systems.

While the BitTorrent mechanism seems to be well designed to support file sharing applications its extension to a real-time distribution is still an open issue. This research aims to the design of a P2P real-time video distribution system using the BitTorrent protocol as its P2P basis. On a first analysis, BitTorrent seems not to be appropriate for being used as basis for a real-time video distribution system, due to the fact that it has been designed for the distribution of large static files and not being equipped with any kind of mechanism capable of the distribution of a continuously incremented bit stream, like those found in real time video distributions. Additionally functionalities like the rarest-first prioritization criteria used in the request for piece files, seem to make BitTorrent even more unappropritate, since their characteristics are not chronologically oriented as a video bit stream should be, originating an almost random receiving of the pieces on the downloading peer.

Another challenging problem could be the considerable startup latencies verified by P2P systems in general. This well known issue has been targeted in several studies, like in [6], which similarly to this research, considers a P2P system to deliver scalable video to peers.

After a detailed analysis of the BitTorrent and due to the simplicity of this protocol, it would be of great interest to investigate how it can be adapted to support a real-time distribution of scalable video. This is the aim of the next chapters.

Chapter 4

Transmission of H.264/SVC Streams over IP Networks

The aim of the work described in this chapter is to analyze how different technologies including scalable video coding, the associated file structures and formats, distinct transport protocols together with a BitTorrent overlay could be combined to form an Application Layer point-to-multipoint distribution of video. To achieve that objective this chapter presents and describes the initial steps performed in terms of design and implementation of a system capable of transmiting H.264/SVC using a BitTorrent Peer-to-Peer system over the Internet.

4.1. Adaptation of H.264/SVC Streams for Transmission over IP Networks

As previously explained, in H.264/SVC, each NAL unit has a direct association with three basic scalable dimensions: spatial (i.e., resolution), temporal (i.e., frame rate) and quality (i.e., SNR), which in turn can be identified using three identifiers: dependency_id, temporal_id, and quality_id. These identifiers can also be referred as a (D, T, Q) tuple, as described in [11].

4.2. Fragmentation and Reassembling of H.264/SVC Streams

In order to implement a BitTorrent P2P point-to-multipoint distribution of H.264/SVC streams, we started by encoding four different video sequences using the H.264/SVC encoder [14], with two image definitions (424x240 and 848x480), four levels of temporal scalability (30, 15, 7.5 and 3.75 fps) and two levels of SNR quality.

In a first stage, the H.264/SVC bit stream needs to be fragmented in several chunks. Different from tipical BitTorrent chunks, in which all chunks have the same size, in this case the tipical variable bit rate of the encoder requires a different solution. Accordingly, the chunk size was made equal to 2 seconds of video playback time, leading to different chunk sizes.

Chunks are afterwards fragmented in several transmission layers, according to Network Adaptation Layer (NAL) unit types, and in response to the scalable spatial, temporal and SNR layers (i.e., (D, T, Q) identifiers). Table 4.1 represents the (D, T, Q) identifiers of each transmission layer and Figure 4.1 presents the block diagram of the implemented chunk and layer partitioning/reassembling functions.

Transmission Layer	**SVC NAL Unit**		
	D	***T***	***Q***
Layer 0	0	0, 1, 2 and 3	0
Layer 1	0	0	1
Layer 2	0	1	1
Layer 3	0	2	1
Layer 4	0	3	1
Layer 5	1	0, 1, 2 and 3	0
Layer 6	1	0	1
Layer 7	1	1	1
Layer 8	1	2	1
Layer 9	1	3	1

Table 4.1: Transmission layers according to SVC scalability layers.

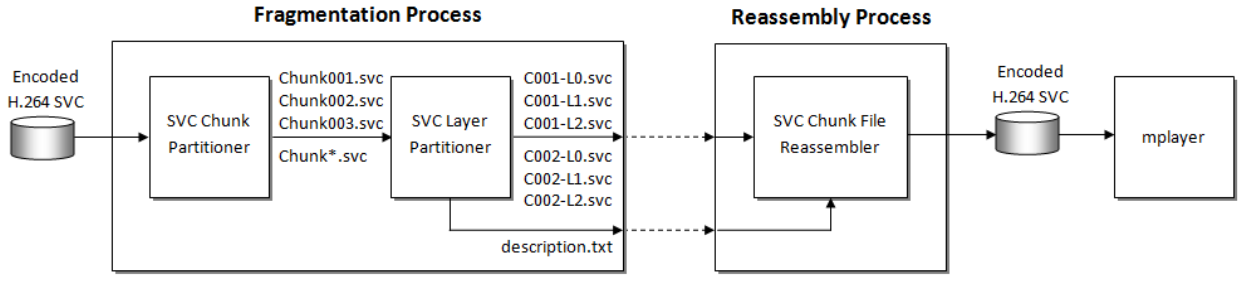

Figure 4.1: Block diagram representing the fragmentation and reassembly processes of the H.264/SVC bit streams.

The first block in Figure 4.1, performs the fragmentation of the original video in chunks of 2 seconds. In order to do it, it analizes the H.264/SVC bit stream searching for NAL units of

Supplemental Enhancement Information (SEI) type, which carry the information about the major video parameters. The delimitation of chunks using these NAL units results from the encoding stage and supports the decoding of each chunk independently.

In the second block, each chunk is fragmented in several layer files, which represent different transmission layers. This is done by analysing the (D, T, Q) tuple of the each NAL unit.

At the receiver, the reassembling of the H.264/SVC bit stream was built in order to obtain a decodable sequence at the receiver, using a subset of the original quality and supporting real time H.264/SVC decoding, as supported by [16]. The real-time requirement also imposed that recently added receivers should be able to start decoding the video at any playback time; i.e., not necessarily at the beginning of a TV program.

During the fragmentation of the original bit stream in chunks and layers an auxiliary description file was created containing the information of the interdependency of the SVC layers in each group of chunk files. This file was meant to be used as reference on the reassembling process.

Since some parts of the bit stream can be lost in the transmission, during the development of the reassembler process it was verified the need for an additional field that references the order of sequence of each NAL unit within the original H.264/SVC bit stream. For this purpose, a new 2-byte field called Sequence Number (SeqNum) was added between the NAL units start code and the beginning of the NAL unit data [1], as represented in Figure 4.2.

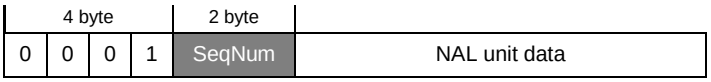

Figure 4.2: Example of the NAL unit structure with the 2-byte sequence number inserted field.

Finally, the reassembler module combines the chunk files, using the interdependency information of the H.264/SVC layers obtained from the description file and reorders the received NAL units using their sequence numbers. As the sequence number of the NAL unit field are only used for the multiplexing the NAL units at the reassembler process, they are discarded after this step.

4.3. Transmission of H.264/SVC Streams Using a Web Server

Based in the previous implementation, in this step a system capable of transmitting H.264/SVC bit streams using the HTTP protocol was implemented. To achieve this, minor improvements had to be made to the system described in section 4.2. The description file coming out of the partitioning process was published online, together with the chunk and layer files of the H.264/SVC bit stream using an HTTP Web Server. Additionally the reassembler process was made to download these layer and chunk files, which were published in the Web Server.

The complete system, capable of transmitting H.264/SVC bit streams over the Internet is shown in Figure 4.3.

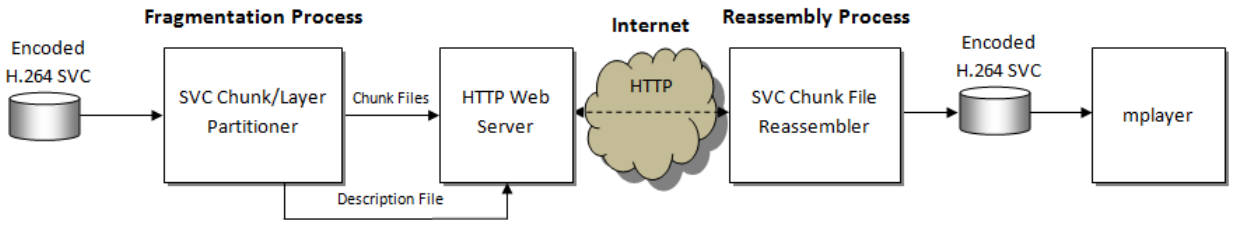

Figure 4.3: Block diagram representing the transmission of H.264/SVC bit streams using an HTTP Web Server.

4.4. Transmission of H.264/SVC Streams Using a P2P Network

From the architectures implemented in sections 4.2 and 4.3, the following step was to implement the capability of transmitting H.264/SVC bit streams over the Internet using a P2P network. For that purpose, as explained in Chapter 3, the BitTorrent P2P file-sharing protocol was chosen.

Given the system obtained in section 4.3, three relevant improvements needed to be made. First, the description file needed to be converted into a torrent file, as required by BitTorrent. For this, all the information of the description file was included in the new torrent file, together with the required URL of the tracker and the fixed piece size of the torrent. Figure 4.4, represents an example of such torrent file.

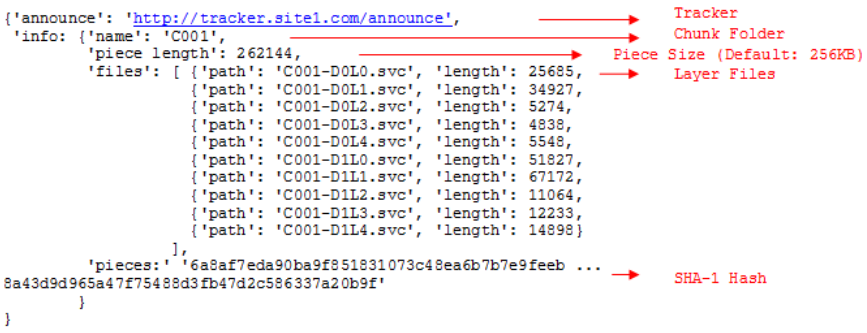

Figure 4.4: Example of a torrent file structure for a H.264/SVC bit stream.

The second change made to BitTorrent was on the piece selection method. In general, the BitTorrent protocol is not suited for real-time applications and a great part of this derives from its piece selection method. It does not respect the chronological order of the events in the bit stream when downloading the required pieces. To try to solve this issue, we tested a sequential piece selection method.

Finally, the third and last necessary improvement was to create a relationship between the torrent pieces over the network and the several transmission layers. For this, the chunk files needed to be padded to match multiples of the fixed piece size defined in the torrent file.

Piece Size	Number of Files/Pieces	Total Files/Pieces Size [kByte]	Overhead
Original Files	100	3484.8	0.0 %
1 kByte	3532	3532	1.4 %
2 kByte	1792	3584	2.8 %
4 kByte	924	3696	6.1 %
8 kByte	489	3912	12.3 %
16 kByte	270	4320	24.0 %
32 kByte	168	5376	54.3 %
64 kByte	122	7808	124.1 %
128 kByte	104	13312	282.0 %
256 kByte	100	25600	634.6 %

Table 4.2: Chunk files after padding to match torrent fixed piece size.

Table 4.2 represents the size of the chunk files after being padded to match each torrent file fixed piece size (between 1 and 256 kByte) and the associated overhead, considering the sizes of each chunk and layer of the video sequences. As the number of pieces increases, the complexity of the

distribution also increases, however as can be verified in Table 4.2, less overhead is obtained as the piece size decreases. Therefore a commitment between complexity and efficiency must be made.

The standard value for pieces is 256 kByte [14], which given the chunks per layer sizes causes an overhead of approximately 634.6%, which is unacceptable. Due to limitations on the minimum value defined for the fixed piece size by the software used for the tests, a 16 kByte fixed piece size was used (yielding aprox. 24.0% overhead).

The padding of the chunk files to match the torrent file fixed piece size, creates the so needed relationship between the torrent pieces and the video transmission layers. These results show that bigger chunks per layer sizes should be used, which can be obtained either by aggregating more information in one layer or using higher definitions.

4.4.1. Implemented Application

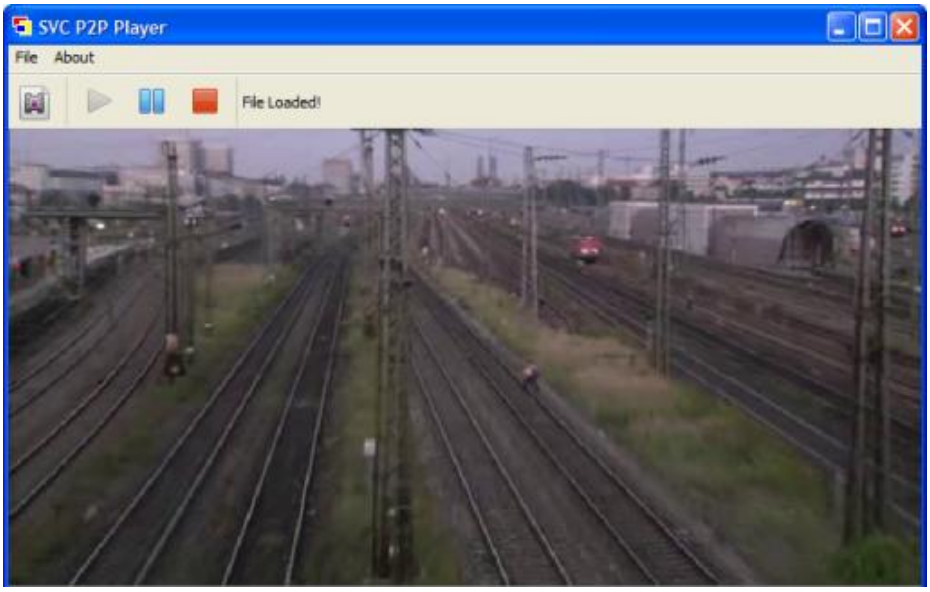

Figure 4.5: Screenshot of a software developed for testing the transmission of H.264/SVC over P2P networks.

Figure 4.5 shows a screenshot of the software developed, using Perl programming language, for testing the transmission of H.264/SVC over P2P networks. The distribution of the video content consisted on a computer with connection to the Internet seeding all the chunks of the torrent file available through the execution of a common BitTorrent client. In the receiver side, the torrent file was given as input to the receiver process. As each peer registered in the swarm and after a few

seconds the download of the pieces of the chunk files started in a sequential order. The receiver reassembled the H.264/SVC bit stream, which was then reproduced.

4.4.2. Resulst Analysis and Identified Challenges

The defined system was able to deliver the scalable video using a BitTorrent P2P overlay. However, in this stage several challenges were still unfulfilled.

First of all, the system lacked on reliability. As a result of that, several performed tests were only capable of receiving and reproducing parts of the bit stream.

Also the torrent file, as it was specified, requires the content to be completely available before the startup of the transmission. Although it could be adjusted to work properly with pre-recorded and stored content, as considered in [7], it still would not be oriented to support a real-time encoding, as considered in the following.

Additionally, the system took in some cases almost a minute to start receiving content by some peers. Once more, besides working properly and performing a successful transmission, the values of delay in the system do not meet the requirements for a real-time video distribution system in terms of the delay between channel switching and the start of video reproduction. Moreover, since these tests were performed using several peers in the same geographical region, they have shown that the signaling process during the P2P startup plays a very important role in the final delay. In fact the reduced number of hops between peers was not enough to assure short delays.

It was also verified that the sequential piece selection method wasn’t appropriate. Although the results obtained were better than the BitTorrent native rarest-first piece selection method, the absence of layer prioritization criterias resulted in an unexpected waiting for pieces from higher enhacement layers, instead of giving a higher priority to the chunks closer to playback. At least the lower transmission layers should always be delivered in time to receivers for their reproduction. This feature should avoid the complete loss of certain chunks with the corresponding subjective quality degradation.

Finally, the implemented environment was extremely difficult to test with a large number of terminals. The requirements for more equipments, combined with different capabilities considerably increases the implementation complexity, due to the need of setting up, customize and configure those equipments. Moreover, the comparison of different P2P parameters and the optimization of these parameters require the implementation of a controlled environment.

4.5. Summary

As described in the last section (section 4.4), the implemented system was able to deliver the scalable video using the BitTorrent P2P overlay. However, the lack of reliability of the system, a startup delay too high and the requirement that the download file should be completely available before the transmission startup require a modification in the P2P system that permits the investigation of amore appropriate piece selection method, in an controllable and reproducible environment.

In order to reduce the overhead introduced by fragmentation of the chunks into pieces, the choice of an adequate torrent fixed piece size is very important. Also several cautions could be taken at the H.264/SVC bit stream encoding stage, as for instance the generation of adequate sized chunk files to be padded. Ideally, these chunk files should have the size of exact multiples of the torrent fixed piece size or slightly smaller than the nearest upper multiple of the torrent fixed piece size, avoiding large volumes of padding bytes.

In order to overcome all those challenges, an optimization of the P2P distribution of scalable video will be implemented in the next chapters.

Chapter 5

A BitTorrent Simulation Framework

In the following part of this study, a simulation framework is implemented based in the BitTorrent operation. Its main objective is to obtain a testing environment capable of simulating the behavior of large swarms, so the changes performed to the BitTorrent could be tested in a controlled environment.

While the results of Chapter 4, were all based on experiments using real software and real computers connected to the Internet, the vast amount of variables used in such a P2P live distribution, combined with a high number of peers for larger swarms, can more easily be evaluated using a simulation framework.

In this chapter, combined with the description of implemented the BitTorrent simulation framework, we also describe several tests performed and the obtained results are analyzed and discussed. The aim of the simulation is to optimize the real-time transmission of scalable video using a P2P network, when combined with a real-time encoding of H.264/SVC.

5.1. Simulation of Real-Time H.264/SVC using BitTorrent

The simulation framework was implemented using a discrete event simulator that was developed using Matlab, in accordance with the BitTorrent protocol specification [14]. Among other features it included: the announce procedure between peers and the tracker of the swarm; the information exchanges between peers regarding their different interests, the content availability and the requests for download of the piece files; together with the choke/unchoke mechanism of each peer in the swarm.

For this environment to work as expected, an independent entity within a group of several pre-defined objects and variables was created in each peer, that translates: the upload and download

available bandwidth; the neighbor peer list and the associated requests for pieces of each of these neighbor peers; the status of the choke/unchoke mechanism for each connection; and the entrance/exit instants in the swarm.

During the execution of the simulated tests, the values of variables were continuously changed and processed. The results obtained for each of these tests reflect the sampling of the different values of these objects and variables for a set of simulations.

During this study the framework suffered several improvements and changes resulting from the introduction of new features resulting from tests; however the compatibility of its behavior with the initial specification of BitTorrent was always taken into account. It resulted in the implementation of an accurate simulation framework capable of a strict reproduction of the BitTorrent protocol behavior as described in [14]. This model enabled the adjustments of variables like chunk and layer priority and a faster measurement of the results, using a much higher number of terminals.

Given the characteristics of BitTorrent described on Chapter 3, some improvements needed to be made to the protocol in order to use it as support of a real-time transmission of H.264/SVC bit streams. In the following we describe each of them.

5.2. Incremental Torrent File Information

Since the BitTorrent protocol was developed with the purpose to be used for file-sharing, the torrent file is a static text file containing the whole structure of the encoded file and the information required for downloading the content files. In a real-time encoding and transmission of video however, as new H.264/SVC chunk files are generated by the source peer, a mechanism is required to inform other peers within the swarm, about where these new files are located.

To keep track of these changes, peers should have access to a simple content update mechanism, similar to the one available in a Really Simple Syndication (RSS) feed. A RSS feed is a standardized technology to directly access to subscribed contents on websites, allowing users to access only new published content without having to manually inspecting all of the websites they are interested in. It works by requiring the access to a standardized eXtensible Markup Language (XML) file that the RSS reader software requests periodically, searching for updates.

In the following, the usage of a RSS feed was considered to support the periodic update of information about the availability of new chunk files among the peers of the swarm.

For this purpose, we considered the main server of the torrent, in which the video chunk files are generated, as the same server responsible for managing and update the RSS feed XML file for every new generated chunk. In order for the peers in the swarm to be aware of the availability of those new chunks for download, they should periodically query the RSS feed in the server for updates. A new parameter should be included in the information on the torrent file with the location of the RSS feed.

5.3. Sliding Window Piece Selection Method

Since the rarest-first piece selection method used by the BitTorrent protocol is not suited for real-time applications and a purely sequential piece selection method seems not to be the best option in terms of real-time, as verified in chapter 4; a new piece selection method was considered in the following tests.

A sliding window based solution, as the one proposed in [17], seems to be more appropriate, as some layers could be dropped after a timeout, in case they are not received. Therefore a solution similar to the method described in [17] was chosen, with the introduction of some improvements to support the scalability properties of an H.264/SVC bit stream.

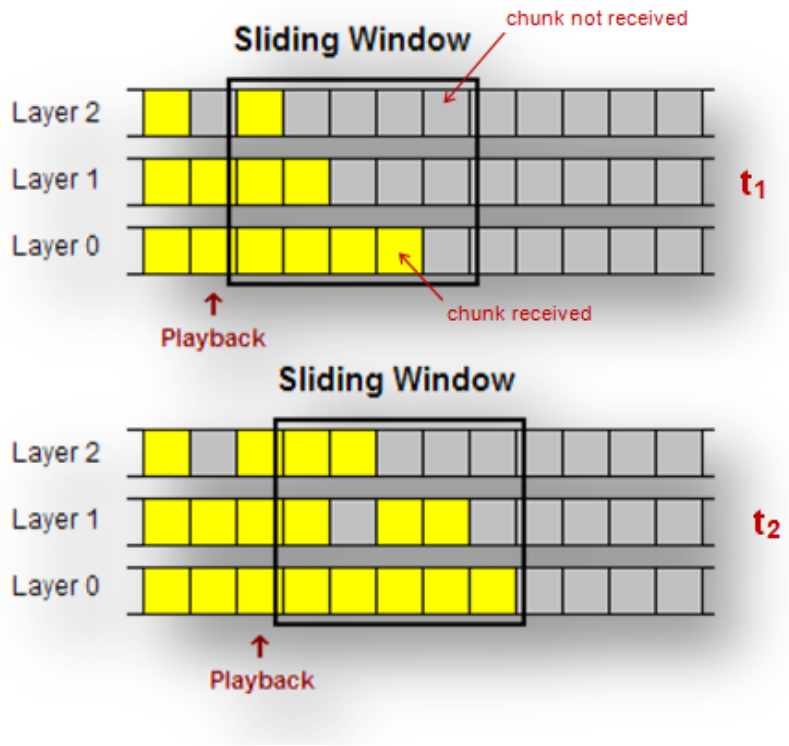

Figure 5.1: Example of a sliding window piece selection method for an SVC bit stream with 3 scalability layers.

Figure 5.1, shows an example of the sliding window selection method. The window contains the next N chunks/pieces of the different layers needed to reproduce the H.264/SVC bit stream in the original events order. Peers can only request pieces inside the sliding window and need to discard requests for pieces of past events.

When using different H.264/SVC layers, peers must decide which chunk and layer to request first. For this purpose a prioritization criteria was defined that combines both the SVC layer index and the remaining time for its reproduction. Based on this, lower layers, and chunks closer to playback time, have been given higher priorities.

In Figure 5.2, an example of such priority scheme can be observed. All chunks and layers outside the sliding window assume the normal value of priority of the BitTorrent protocol (priority=2), while the priorities of pieces inside the sliding window increase according to the importance of the SVC layer (i.e. base layer is the most important layer) and decrease according to the chunk number.

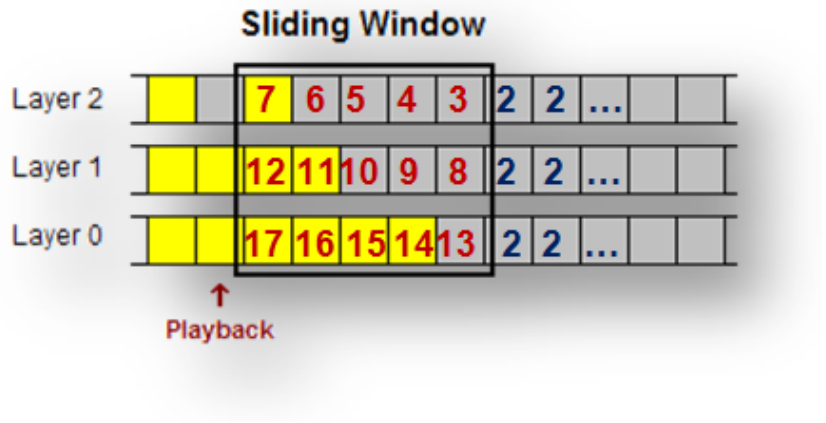

Figure 5.2: Example of the chunk prioritization criteria according to the importance of the SVC layer and the chunk number.

In this solution, the size of the sliding window seems to be an important feature. By either reducing or enlarging the window sizes peers can respectively request a higher number of chunks and layers, or concentrate in obtaining a lower number of chunks and layers. A larger sliding window makes sparser the focus of each peer in getting those chunks that are closer to playback time, wasting bandwidth requesting chunks that are not so important. On the contrary a thin sliding window makes peers concentrate in receiving important chunks, but approximates the distribution to the sequential piece picking solution.

In the following we have tested several window sizes for the prioritization mechanisms represented in Figure 5.2.

5.3.1. Simulation Results

As previously described in sections 5.2 and 5.3, the BitTorrent protocol requires some improvements to be used in real-time video distribution. Those features include the incremental update of the torrent file and the sliding window piece selection method. Both of these features were introduced into the simulation model with the objective to test their behavior when compared with the normal BitTorrent for SVC distribution over large swarms.

The following tests were performed considering a BitTorrent swarm with 100 peers during the transmission of an encoded H.264/SVC video sequence with 900 seconds (15 minutes) of duration. This H.264/SVC video sequence considered in the simulations was already used in the tests of Chapter 4. It had an average bit rate of 1.45Mbps and was fragmented in 10 transmission layers (as described in section 0).

All peers in the swarm had a maximum uplink rate of 2.0 Mbps and a maximum downlink rate of 20.0 Mbps. The maximum P2P upload rate of peers was limited to 90% of the uplink rate (i.e. 1.8Mbps). During simulations all peers entered the swarm in a random distribution along the first 30 seconds of the experiment. Tests were repeated 3 times for each of the four values of the sliding window sizes, namely: 2, 3, 4 and 5 chunks.

In the following tests, chunks that are not available do not stop the playback of the video nor the sliding window. Similar to a broadcast like distribution, after connecting to the swarm, peers try to decode the video continuously. However when the peer is not able to receive a chunk in time, the image halts during that period, jumping to the next received chunk. In the next sub-section we will consider an alternative to this method.

Besides content availability, another important metric of quality for real-time distributions is the delay between content availability and playback, which is described in Figure 5.3. As video chunk files are continuously generated, they become available in the server which acts as the initial seed of the swarm. As each new peer connects to other peers in the swarm, the delay between content generation and playback depends on the set of peers it selects. Regarding Figure 5.3, as peers A, B and C start receiving chunks from the correspondent sliding window they start playing the video stream. The difference between the instant of time in which the encoded H.264/SVC chunk files are created in the server and the instant the video stream is reproduced at each peer, defines the delay between the server and the playback of the stream at that peer.

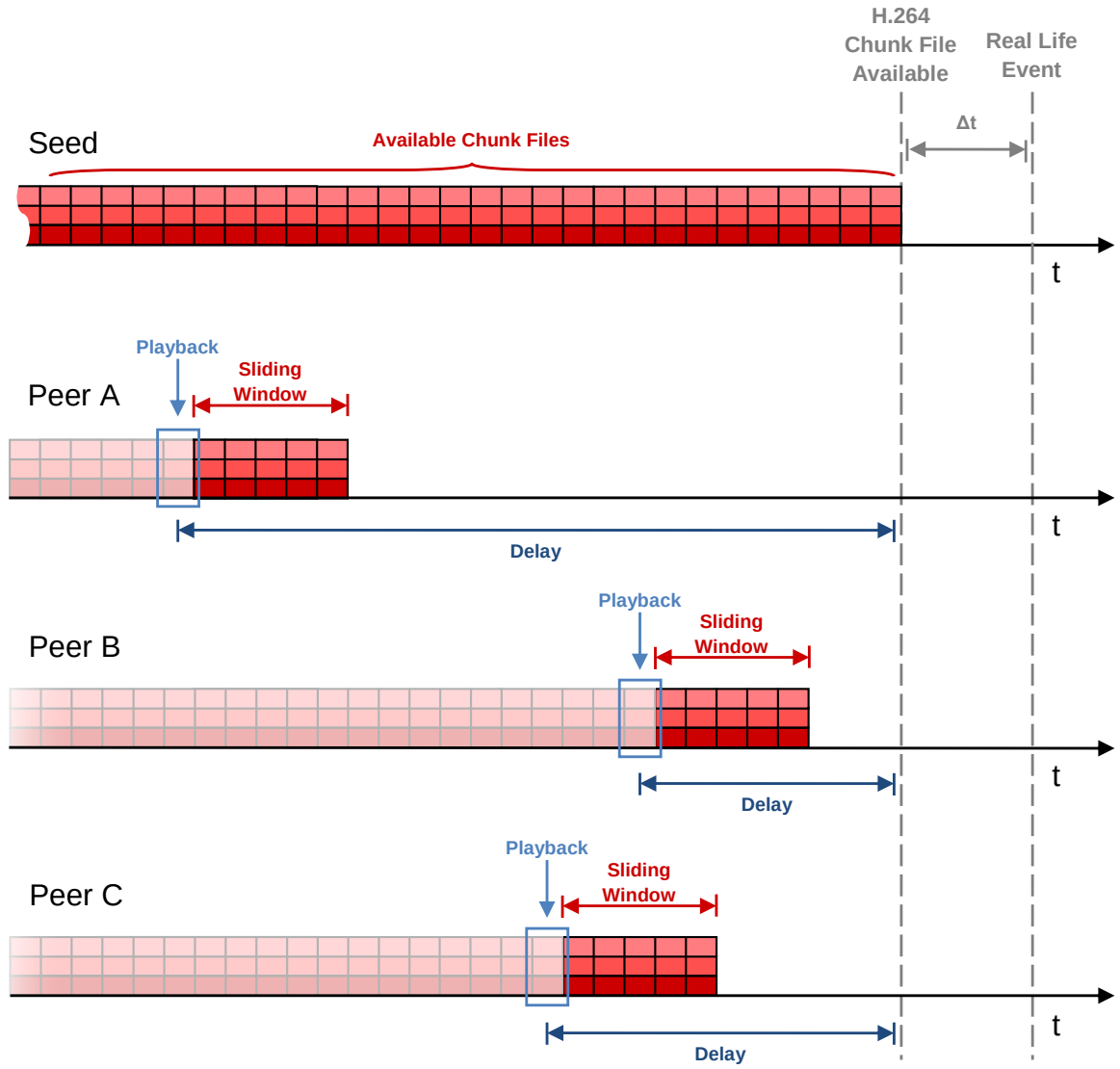

Figure 5.3: Delay between the availability of the chunk files in the server and the playback of the stream on the receiving peers.

Given these considerations, Figure 5.4, presents the results obtained in the simulation tests regarding the evolution in the number of transmission layers received by the peers in the swarm as the chunks are being generated, comparing different window sizes. It can be verified that the best results were achieved for sliding window sizes of 4 and 5 chunks. While window sizes of 2 and 3 chunks guaranty that a higher number of receivers are able to get higher layers, they present the drawback of a significant percentage of peers not being able of receiving any layers during some period of time.

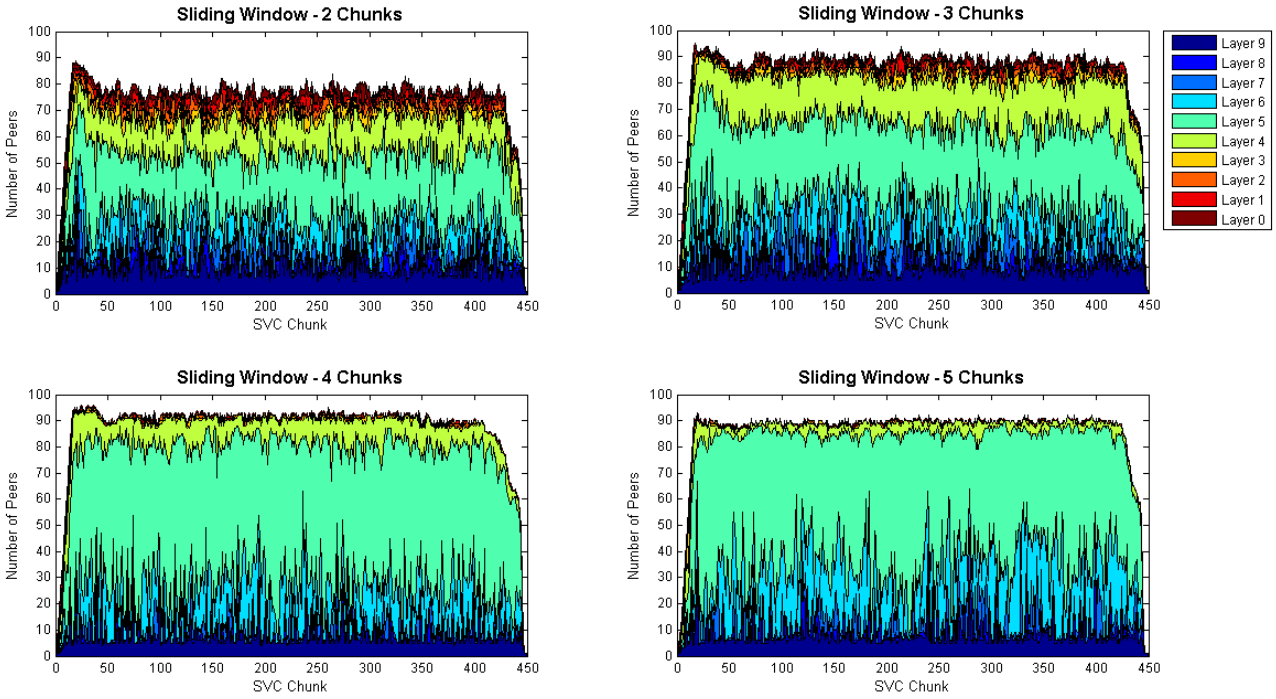

Figure 5.4: Evolution of the number of transmission layers received using different sliding window sizes: 2, 3, 4 and 5 chunks.

Table 5.1 presents the average number peers receiving each of the transmitted layers. It can be verified that the number of peers receiving 6 layers increases from 53.4% to 85.3%, when varying the sliding window size from 2 to 5 chunks and that the number of peers with chunk gaps decreases from 22.4% to 10.4% respectively.

	Sliding Window Size			
	2 Chunks	***3 Chunks***	***4 Chunks***	***5 Chunks***
None	22.4%	11.1%	8.6%	10.4%
Layer 0	77.6%	88.9%	91.4%	89.6%
Layer 1	75.3%	87.6%	91.2%	89.5%
Layer 2	72.8%	85.8%	90.8%	89.3%
Layer 3	69.9%	84.0%	90.4%	89.2%
Layer 4	66.7%	81.6%	89.8%	88.9%
Layer 5	53.4%	64.4%	80.9%	85.3%
Layer 6	28.7%	32.5%	26.7%	34.6%
Layer 7	18.6%	19.5%	11.8%	14.4%
Layer 8	12.9%	12.6%	8.5%	9.8%
Layer 9	9.4%	9.0%	7.1%	7.5%

Table 5.1: Ratio of transmission layers received using different sliding window sizes: 2, 3, 4 and 5 chunks.

Given the delay computed as described in Figure 5.3, Figure 5.5 presents the histogram of delays between the creation of the chunk files in the server and the playback of the video stream on the receiving peers, for different sliding window sizes of 2, 3, 4 and 5 chunks.

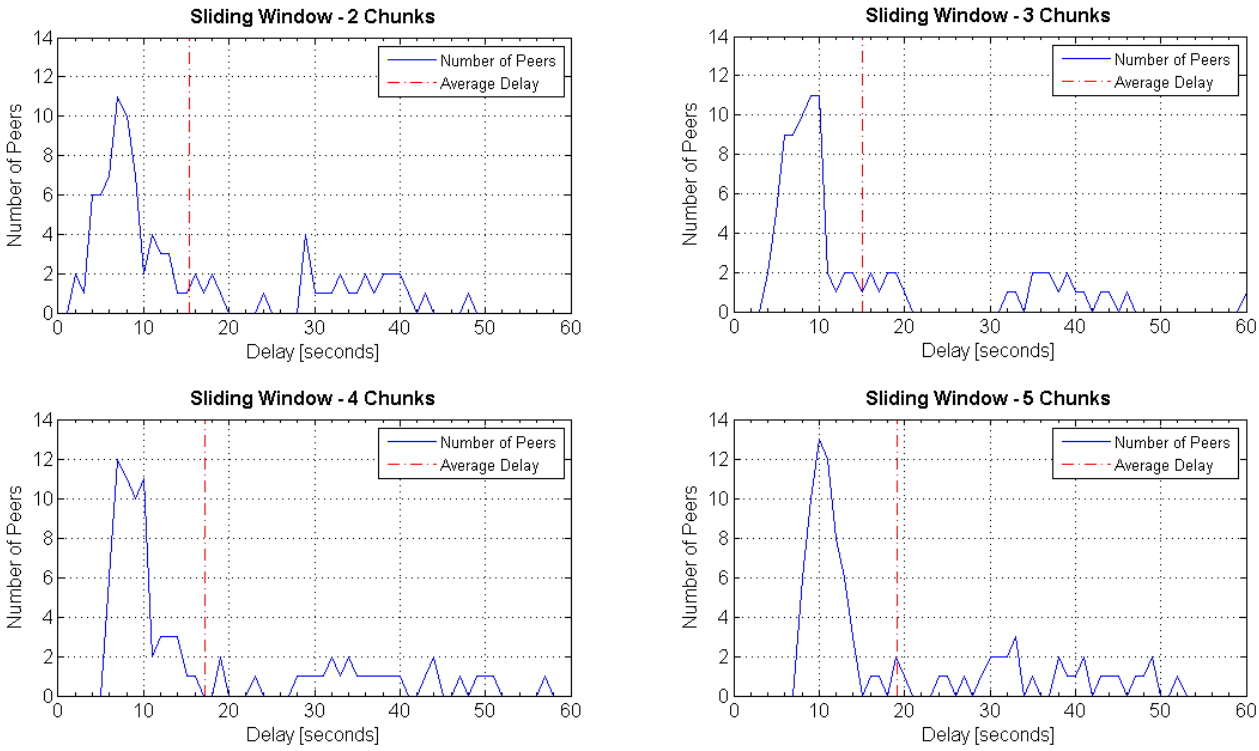

Figure 5.5: Histogram of delays between the creation of the chunk files in the server and the playback of the video stream on the receiving peers.

It can be verified (in Figure 5.5) that the best result was achieved for a window size of 3 chunks, which has yielded an average delay of nearly 15 seconds. It can be verified by the histograms that the delay introduced by the sliding window size, sums up with the delay of the distribution itself, because initially the playback only starts after the filling of the sliding window with chunks of at least the base layer.

Globally histograms demonstrate a high dispersion in delays. While a considerable ratio of peers are able to get the video with a delay lower than 20 seconds, nearly 25% of the other peers are subject to significantly higher delays.

5.3.2. Summary

The tests performed using the simulation model with the prioritized sliding window have shown that a window sizes of 3, 4 or 5 chunks are capable of delivering at least one layer to nearly all

receivers. In terms of the delay between the server and the start of video playback, a window of 3 chunks has been capable of reducing its value to nearly 15 seconds. Both results show that, given the conditions defined for these tests, the adjustment of the window size influences the real-time behavior of the distribution.

These results have also demonstrated that there are still several improvements that need to be performed to the system here described. Not only the number of chunk gaps should be reduced but also the number of layers for each peer may be improved.

5.4. Video Buffering Techniques

The results from the tests in the last section, demonstrated that improvements need to be made to the BitTorrent system to be able of using it as a real-time video distribution system.

In the following tests, two different chunk buffering mechanisms will be tested using the simulation framework, namely: Deferred Playback and Initial Chunk Selection Buffering.

5.4.1. Deferred Playback

While in previous section we have considered that the sliding window shifts independently of the available chunks and also that outdated chunks are not requested nor used in the playback, in this section we consider that the sliding window and the playback of the video only happens when at least a base layer chunk is available. This implies that peers that are not able to download chunks in time will defer their playback. At the player the video will halt waiting for missing chunks and when they arrive it will continue at the same point where it was, i.e. without time gaps. In terms of P2P distribution however, this implies that a peer that performs such defer, in both sliding window and playback, will increase its temporal distance from the seed, i.e. video source.

This feature enables the H.264/SVC software client to wait for the arrival of the missing chunks of the base layer and ensures the complete reproduction of the bit stream a few instants later when the base layer chunk files become available. The absence of this feature in the simulation model used in section 5.3, originated an intermittent reception behavior of chunks in some of the peers resulting in the lack of reliability on the reception of the bit stream.

In Figure 5.6, is shown an example of the SVC base layer buffering mechanism in action, guaranteeing the complete reproduction of the bit stream, stopping and advancing it according to the needs for downloading chunk files. It exemplifies the progress of a peer receiving chunk files

and reassembling them to reproduce the video stream. For t=t1, the peer is downloading chunk files from the defined sliding window and simultaneously reproduces the content of the bit stream as the chunk files become available. In t=t2, the peer continues to download the chunk files, as the playback of the bit stream continues. The same behavior is observed in t=t3, with the peer continuing to download the requested chunk files as bit stream playback continues.

Therefore, for t=t4, the following base layer chunk file has not been downloaded in the expected time compromising the continuity of the bit stream playback, but in order to ensure its continuity without losses, the playback of the bit stream is stopped (i.e. paused) until the chunk file becomes downloaded. Finally, for t=t5, the missing base layer chunk file has already been downloaded and the playback of the bit stream continues.

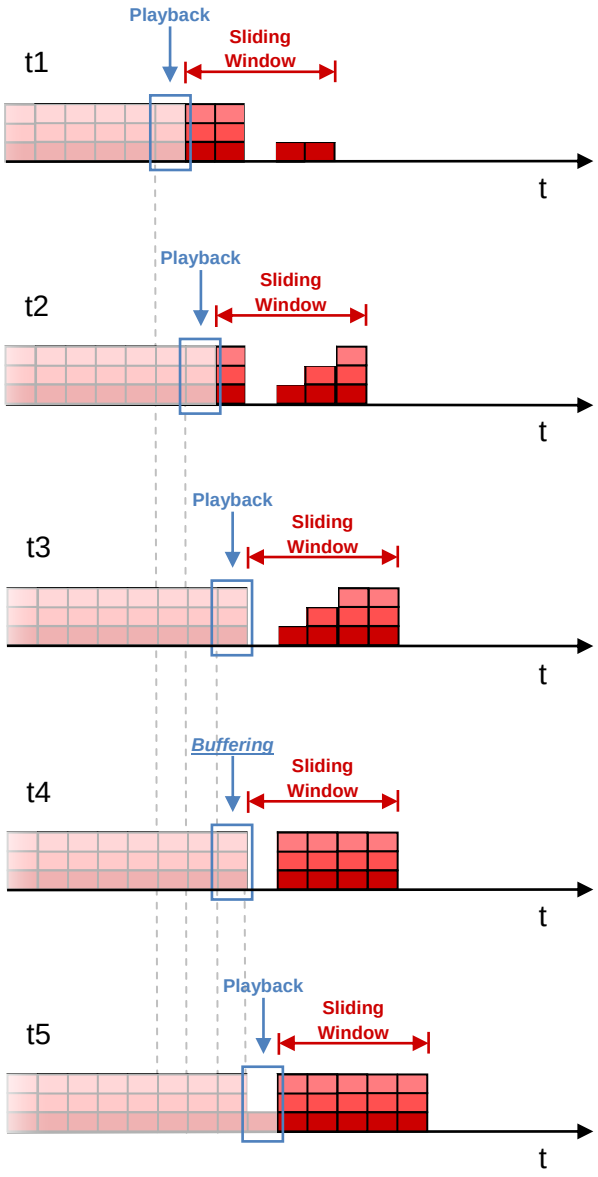

Figure 5.6: SVC base layer buffering mechanism.

5.4.2. Initial Chunk Selection Buffering

The advantage of using a P2P overlay for real-time distribution of content is that each peer can cache chunks that it has already played, but that other peers might still need. The advantage of such solution is that it not only supports the real-time distribution, but also supports program rewinding or the playing of a program at its beginning, without or with a minor intervention of the seed.

At a minor scale, this can also be used to support a P2P distribution where peers do not need be receiving the most recently generated chunks. In fact users tend to be more tolerant to the delay between content encoding and playback than to high values of delays between content request and playback (so called channel switching times). In IPTV standards [18] such channel switching delays should be kept under 2 seconds. In our case, given that we are considering chunks of 2 seconds and that we have window sizes of several chunks we must find a way of quickly fill the sliding window and start playback as soon as possible. In this case, since we are using SVC, we could download the base layer chunks from peers that have cached them, even if this implies starting the video playback at a past time.

This mechanism consists in a sequential chunk file buffering process performed backwards from the initially chosen chunk file to the older ones that are available. In this task, a recently joined peer tries to get a continuous group of base layer chunk files, with the size of a sliding window. This process is repeated continuously until the sliding window is found. After that, the peer starts the playback of the H.264/SVC bit stream from the oldest chunk to the most recent ones and advances subsequently every time a new chunk is reproduced.

5.4.3. Simulation Results

This section describes the tests performed using the simulation model with the buffering mechanisms described in sections 5.4.1 and 5.4.2.

In order to compare these solutions with the results previously obtained, the following tests consisted in using the same simulation model as in section 5.3, but with the introduction of the buffering mechanisms. Given this purpose, the same H.264/SVC encoded sequence and the same BitTorrent like environment were used, monitoring a swarm of 100 peers during the transmission of 900 seconds of the encoded bit stream, with an average bit rate of 1.45Mbps, fragmented in 10 transmission layers (as previously described in section 4.1).

The network capabilities of the peers were the same; all the peers in the swarm had a maximum uplink rate of 2.0 Mbps and a maximum downlink rate of 20.0 Mbps. The maximum P2P upload rate of peers was limited to 90% of the uplink rate (i.e. 1.8Mbps) and during simulations all peers entered the swarm in a random distribution along the first 30 seconds of the experiment.

Each test was repeated 5 times, for each of the different values of the sliding window sizes considered, namely: 3, 4 and 5 chunks.

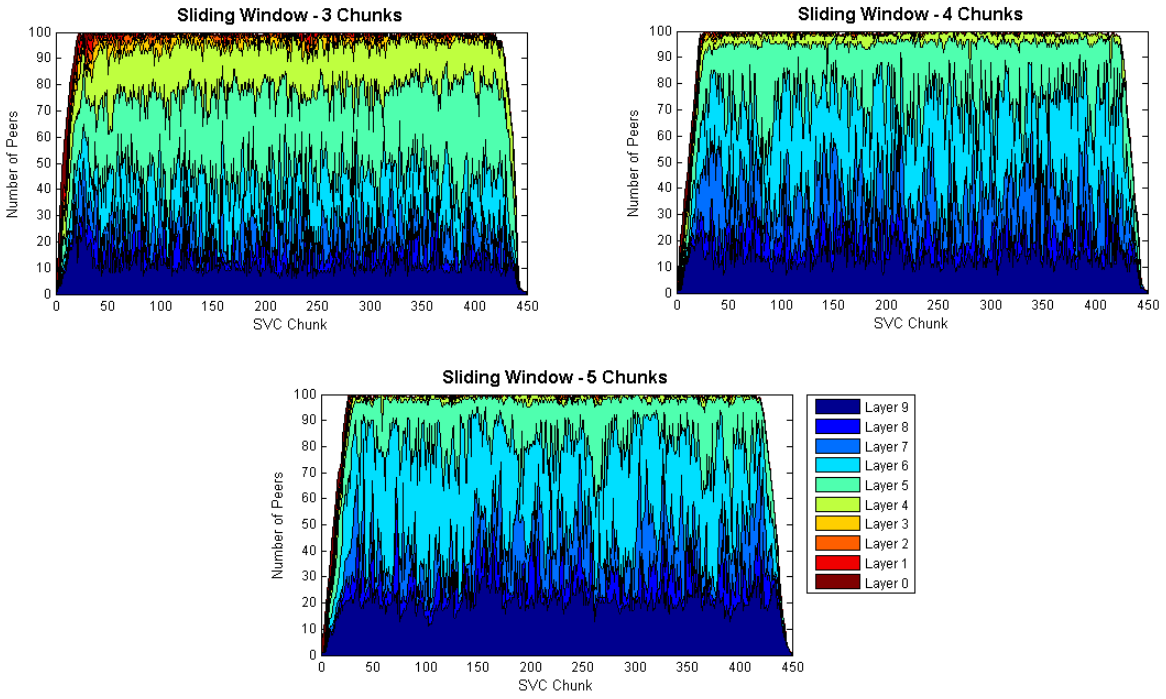

Figure 5.7: Transmission layers received using buffering techniques for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.7 shows the evolution of the number of transmission layers received by the peers in the swarm as chunks are being transmitted. The highest number of receivers able to get higher layers was obtained for the window size of 5 chunks.

This result can also be verified in Table 5.2, where the average number of peers receiving each of the transmission layers is presented. For instance it can be verified that the number of peers receiving at least 6 layers increases from 77.4% to 96.4% when varying the sliding window size from 3 to 5 chunks. Negatively, the number of peers not receiving any layer, more than tripled from 0.4% to 1.3%.

	Sliding Window Size		
	3 Chunks	***4 Chunks***	***5 Chunks***
None	0.4%	0.9%	1.3%
Layer 0	99.6%	99.1%	98.7%
Layer 1	99.2%	99.0%	98.6%
Layer 2	98.3%	98.8%	98.4%
Layer 3	96.9%	98.5%	98.2%
Layer 4	94.3%	98.0%	98.0%
Layer 5	77.4%	94.6%	96.4%
Layer 6	45.5%	72.2%	79.5%
Layer 7	27.1%	40.8%	44.2%
Layer 8	17.4%	22.7%	29.9%
Layer 9	11.8%	14.7%	21.5%

Table 5.2: Ratio of transmission layers received using buffering techniques for different sliding window sizes: 3, 4 and 5 chunks.

Comparing the results in Table 5.2 with the results obtained for the simulation model used in section 5.3 (i.e. without buffering mechanisms, in Table 5.1), a significant increment was verified in the number of peers receiving at least 6 layers. It increased 20% for the sliding window of 3 chunks, from 64.4% to 77.4%; 17% for the sliding window of 4 chunks, from 80.9% to 94.6% and 13% for the sliding window of 5 chunks, from 85.3% to 96.4%.

For the number of peers not receiving any layers the reduction is tremendous from the results obtained in section 5.3, for the different sliding window sizes, reductions of 96%, 90% and 88% can be verified for sliding window sizes of 3, 4 and 5 chunks respectively.

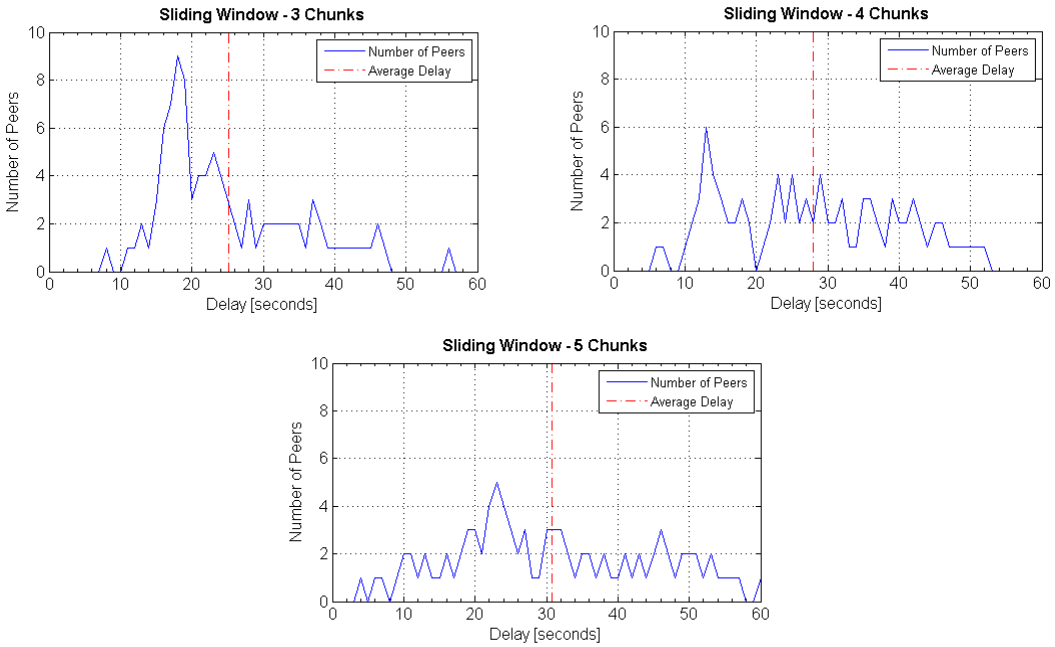

Figure 5.8: Histogram of delays between the creation of the chunk files in the server and the playback of the video stream on the receiving peers using buffering mechanisms

Figure 5.8 presents the histogram of the delays verified between the availability of the chunk files in the server and the reproduction of the stream on the receiving peers. From the results obtained, the best result of this variable was achieved for a sliding window size of 3 chunks, yielding an average delay of nearly 25 seconds. For the sliding window sizes of 4 and 5 chunks, 28 and 31 seconds values where obtained respectively.

In comparison to the results obtained in section 5.3 and considering the best scenario with a sliding window size of 3 chunks, the average delay between chunk availability and playback increases significantly from 15 to 25 seconds.

Figure 5.9 shows the histogram of the time elapsed between the entrance of a peer in the swarm and the instant the video reproduction starts, also described as the "time elapsed until the video playback". The best result of this variable was achieved for a window size of 3 chunks, yielding an average delay of nearly 26 seconds, while for 4 and 5 chunks sliding windows the delay reached 38 and 41 seconds respectively.

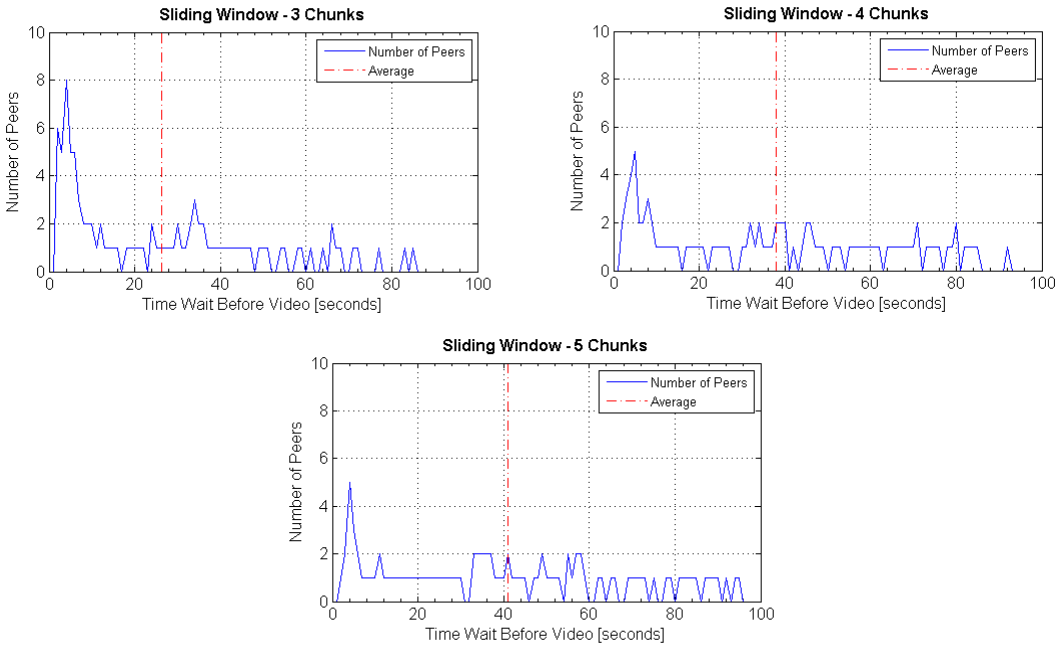

Figure 5.9: Histogram of elapsed time until the video playback using buffering mechanisms

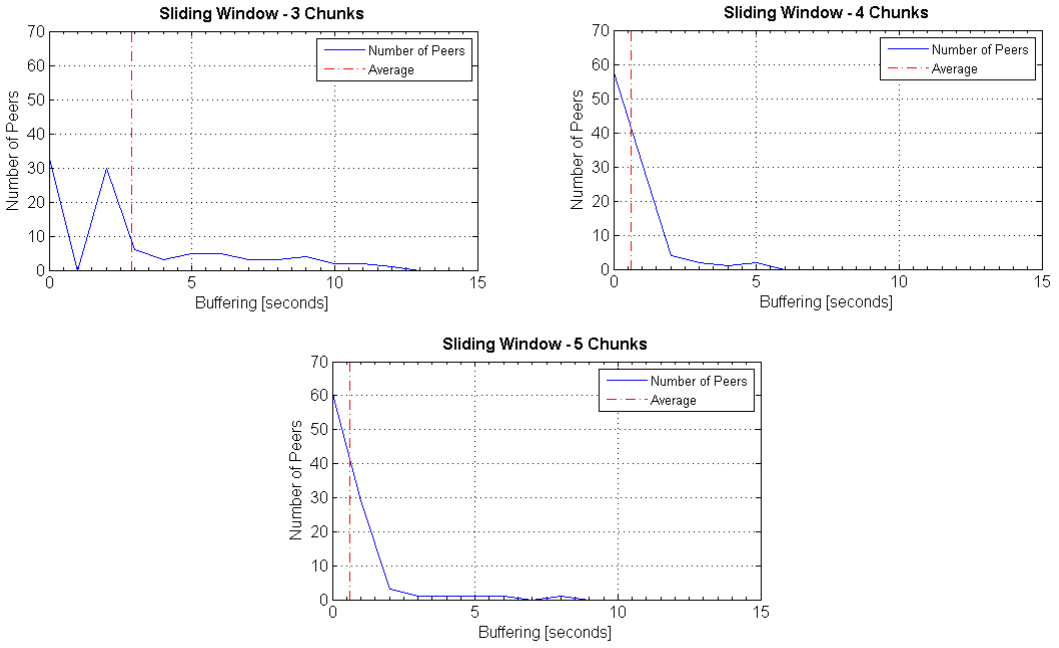

Figure 5.10: Histogram of delays caused by the re-buffering mechanisms

Figure 5.10 shows the histogram of delays caused by pause-and-resume playback of video. These results are a measure of the difficulty that peers have in obtaining chunks in time for video playback.

In this case, the best results were achieved for the sliding window sizes of 4 and 5 chunks, with an average delay of 0.6 seconds. For the sliding window of 3 chunks the delay increased to a value of 2.9 seconds.

5.4.4. Summary

The results obtained show that the introduction of the buffering techniques can yield very positive results, as they increased the number of peers receiving higher transmission layers and drastically reduced the number of peers not receiving any layers. However, an expressive increase of the delay between the availability of the chunk files in the server and the reproduction of the stream on the receiving peers was verified. In terms of the "time elapsed until video playback", the sliding window of 3 chunks has been capable of an average value of 26 seconds, which is unacceptable when compared with common IPTV distributions of video.

These results have demonstrated that, besides the enhancements verified on the number of peers receiving more transmission layers and the reduction of the peers not receiving any layer, there are still several improvements that need to be performed to the system in order to reduce the high values of the several delays measured.

5.5. Optimization of the Number of Downloading Peers and the Initial Choke/Unchoke Interval

The results obtained in section 5.4 are still far from the expected values for a TV distribution system based on BitTorrent. Parameters like the number of transmission layers received by peers, the time elapsed between the creation of the chunk file on the server and its reproduction on the peers, or the delay between a peer entering the swarm and the start of the video playback require further improvements.

Given those requirements, in the following we will try to optimize two important parameters of the standard implementation of the BitTorrent, namely the number of peers downloading from a single source peer and the size of the starting interval of the choke/unchoke mechanism of a peer.

In terms of the number of peers downloading from a single peer, also known as downloading peers, it is expected to influence the number of transmission layers received by peers. In fact, by changing it we are adjusting the probability of finding a chunk/piece file in the swarm, which in turn is expected to change the probability of a peer receiving an answer for a requested chunk file.

Regarding the size of the starting interval of the choke/unchoke mechanism of a peer, it is expected to directly influence the time elapsed between the peer's entering the swarm and the start of the video stream reproduction. In fact, when entering in a swarm a peer starts by having all its connections in the choke state and need to wait for the first choke/unchoke mechanism to unchoke any connection and to permit any exchange of content to start.

This section describes the tests performed with the simulation model previously used in section 5.4 (with buffering mechanisms), analyzing the behavior of introducing variations in the number of downloading peers and in the size of the starting interval of the choke/unchoke mechanism, against the results previously obtained.

Given these objectives, the same H.264/SVC encoded sequence was used (structure and characteristics previously described in section 4.1) and the same BitTorrent environment, using a swarm of 100 peers for transmitting a bit stream during 900 seconds. The network capabilities of the peers were the same; all the peers had a maximum uplink rate of 2.0 Mbps and a maximum downlink rate of 20.0 Mbps, using only a maximum of 90% of the uplink rate (i.e. 1.8Mbps) and during simulations all peers entered the swarm in a random distribution along the first 30 seconds of the experiment.

Each test was repeated 5 times, for each of the three different values of the sliding window sizes considered: 3, 4 and 5 chunks.

5.5.1. Number of Downloading Peers

Figure 5.11 presents the evolution in the number of layers received by the peers while chunks are being transmitted using 6 downloading peers. In this case, the best results were achieved for sliding window sizes of 4 and 5 chunks.

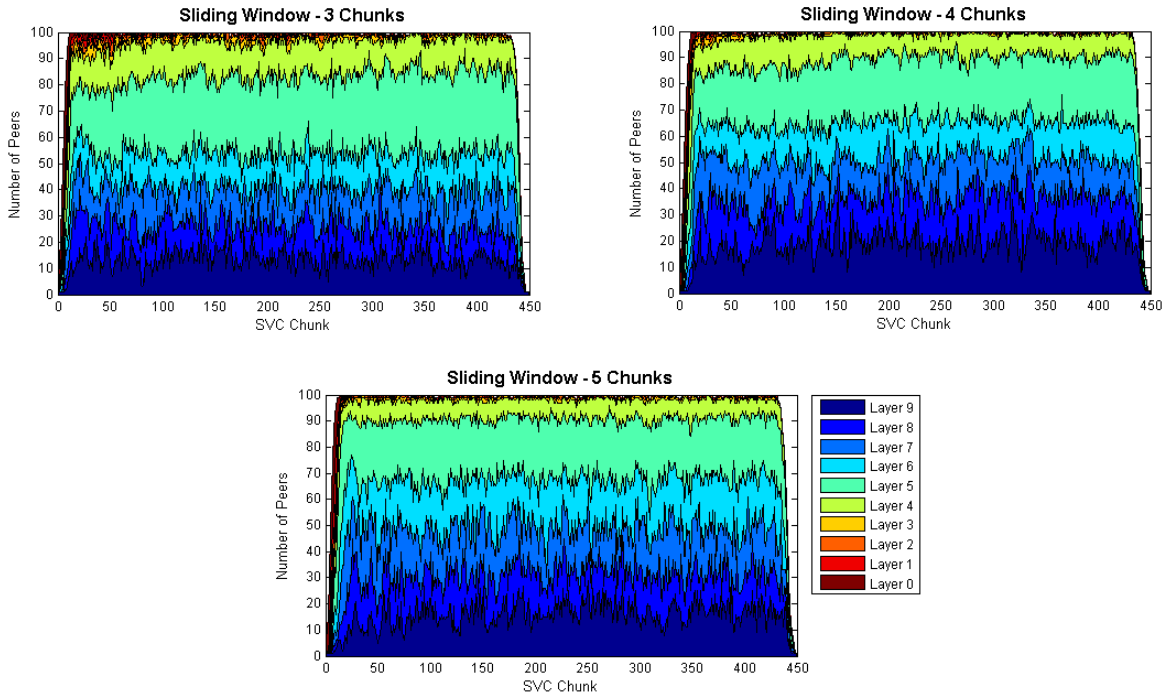

Figure 5.11: Transmission layers received using 6 downloading peers for different sliding window sizes: 3, 4 and 5 chunks.

Table 5.3 presents the average number peers receiving each of the transmitted layers, in which can be verified that the number of peers not receiving any layer disappeared for all the different sliding window sizes.

	Sliding Window Size		
	3 Chunks	***4 Chunks***	***5 Chunks***
None	0.0%	0.0%	0.0%
Layer 0	100.0%	100.0%	100.0%
Layer 1	99.7%	99.9%	100.0%
Layer 2	99.2%	99.8%	99.8%
Layer 3	98.2%	99.4%	99.4%
Layer 4	96.3%	98.6%	98.3%
Layer 5	83.3%	89.3%	90.9%
Layer 6	53.8%	65.2%	67.6%
Layer 7	40.4%	51.3%	48.7%
Layer 8	26.4%	36.5%	31.1%
Layer 9	14.0%	19.8%	16.4%

Table 5.3: Ratio of transmission layers received using 6 downloading peers for different sliding window sizes: 3, 4 and 5 chunks.

When comparing the results in Table 5.3 using 6 downloading peers with the scenario with 4 downloading peers on section 5.4 (Table 5.2), an improvement of nearly 6% (from 77.4% to 83.3%) is verified in the number of peers receiving 6 layers, for a sliding window of 3 chunks. However for sliding window sizes of 4 and 5 chunks, a decrease of nearly 5% is verified, respectively from 94.6% to 89.3% and from 96.4% to 90.9%.

In this case the best result was achieved for a sliding window of 4 chunks, combining a high number of peers receiving at the same time the lowest and the highest layers.

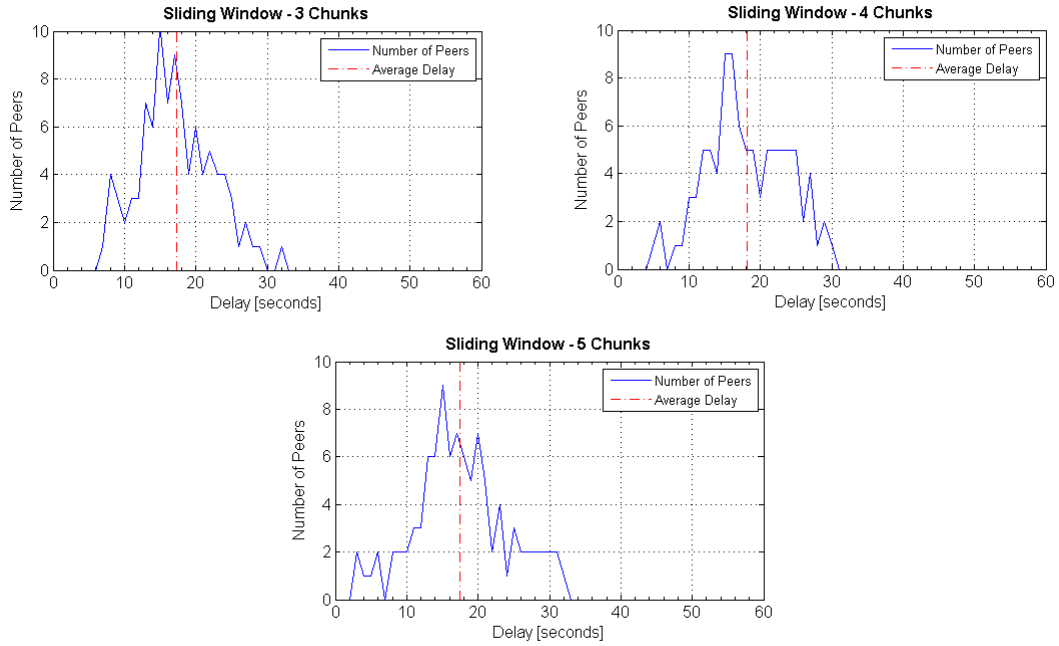

Figure 5.12: Histogram of elapsed time between the creation of the chunk files in the server and the playback of the video stream on the receiving peers using 6 downloading peers.

Figure 5.12 presents the histograms of the delays verified between the creation of the chunk files in the server and the reproduction of the stream on the receiving peers. The best results for this variable were obtained for the sliding window sizes of 3 and 5 chunks, with a delay of nearly 17 seconds. For the sliding window of 4 chunks the result was a delay of 18 seconds.

When comparing the obtained results with the ones obtained in section 5.4, on the scenario with a sliding window size of 3 chunks, the delay decreased from 25 to 17 seconds.

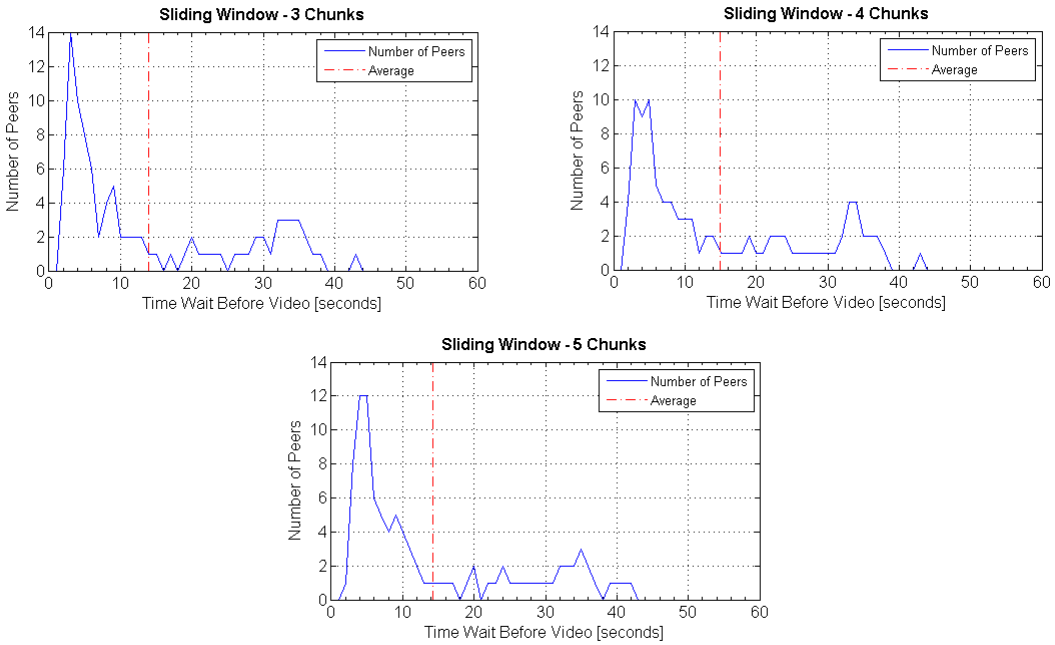

Figure 5.13: Histogram delays from request to video playback when using 6 downloading peers.

Figure 5.13 presents the histograms of the elapsed time until the video playback starts using 6 downloading peers. The best results of this variable were achieved for the sliding window sizes of 3 and 5 chunks, with a delay of nearly 14 seconds; while for the sliding window of 4 chunks was obtained a delay of 15 seconds.

Again comparing these results with the ones obtained in section 5.4, on the scenario with a sliding window size of 3 chunks, the delay has fell almost by half, from 26 to 14 seconds. Although these values continue to be too high, a significant reduction in startup times was verified when increasing the number of downloading peers.

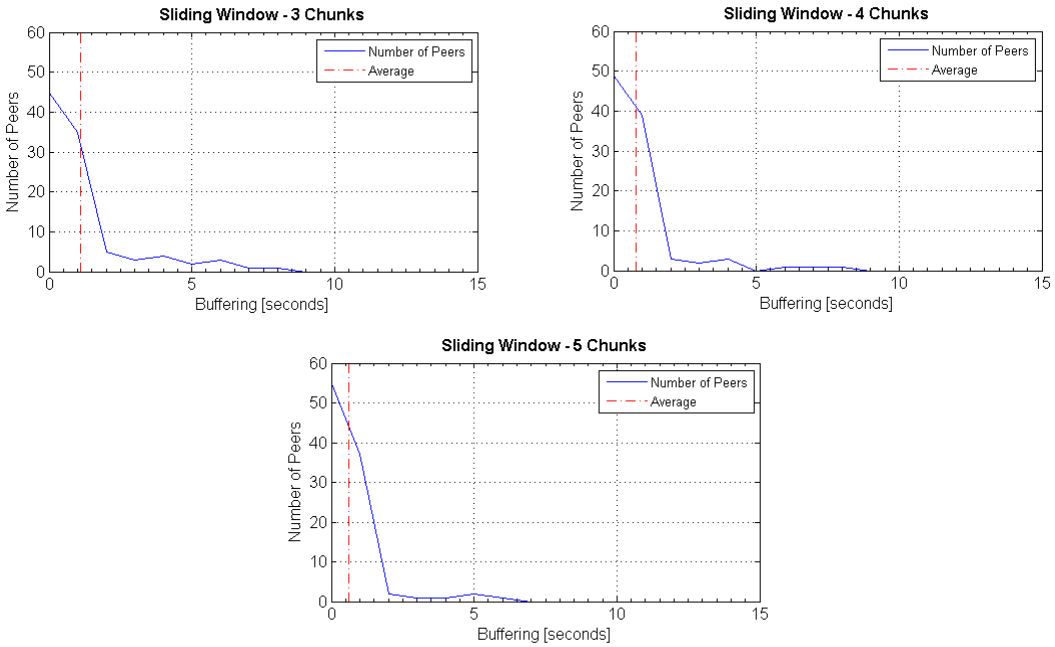

Figure 5.14: Histogram of delays caused by the re-buffering mechanisms using 6 downloading peers.

Figure 5.14 presents the histograms of delays caused by the pause-and-resume playback of video when using 6 downloading peers. The delay values achieved for the sliding window sizes of 3 and 4 chunks were respectively 1.1 and 0.8 seconds, while the best result was achieved for the sliding window size of 5 chunks, with a delay of 0.6 seconds.

In comparison with the average buffering delay values achieved in section 5.4, this value does not change.

5.5.2. Initial Choke/Unchoke Interval

In the following test we will analyze the effect of reducing the choke/unchoke interval from 10 to 5 seconds, keeping the number of downloading peers equal to 4.

Figure 5.15 shows the evolution in the number of layers received by the peers in the swarm as the chunks are being transmitted, when decreasing the initial choke/unchoke interval to 5 seconds. It can be verified that the best results were achieved for window size of 4 chunks. The solution with 4 chunks guaranties that a higher number of receivers are able to get higher layers.

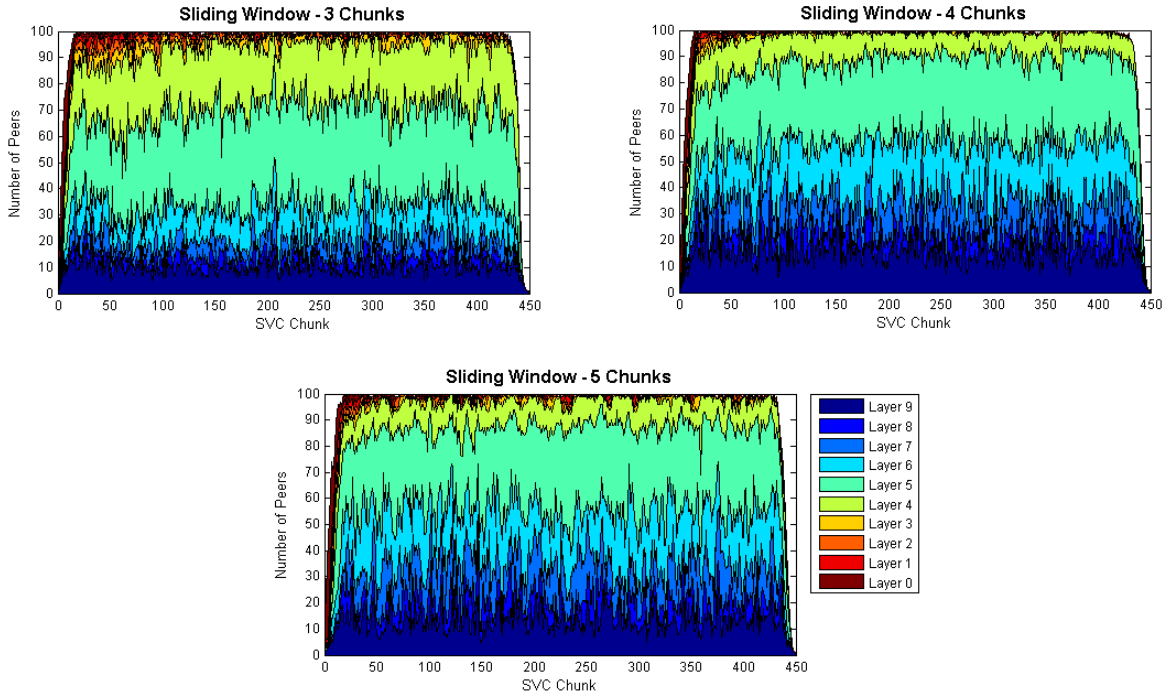

Figure 5.15: Transmission layers received using an initial choke/unchoke interval of 5 seconds for different sliding window sizes: 3, 4 and 5 chunks.

Table 5.4 presents the average number peers receiving each of the transmitted layers, in which can be observed the almost disappearance in the number of peers that at some moment are not able to receive any layer.

	Sliding Window Size		
	3 Chunks	***4 Chunks***	***5 Chunks***
None	0.1%	0.1%	0.1%
Layer 0	99.9%	99.9%	99.9%
Layer 1	99.6%	99.8%	99.6%
Layer 2	98.6%	99.6%	99.1%
Layer 3	97.0%	99.2%	98.2%
Layer 4	93.8%	98.4%	96.9%
Layer 5	69.9%	88.5%	87.2%
Layer 6	33.4%	56.4%	55.6%
Layer 7	20.9%	35.4%	35.1%
Layer 8	14.3%	23.1%	21.3%
Layer 9	10.1%	15.5%	13.6%

Table 5.4: Ratio of transmission layers received using an initial choke/unchoke interval of 5 seconds for different sliding window sizes: 3, 4 and 5 chunks.

When comparing the results in Table 5.4 using a 5 seconds on the initial choke/unchoke timer with the results obtain using the regular choke/unchoke timer (Table 5.2), a decrease trend is verified in the number of peers receiving at least 6 layers for the different sliding window sizes. A decrease of nearly 5% is verified, from 94.35% to 89.14% and from 96.21% to 90.74%, respectively for 3 and 5 chunks. A very significant improvement is verified in the number of peers not receiving any layer, being reduced to 0.1% for all the different sliding window sizes.

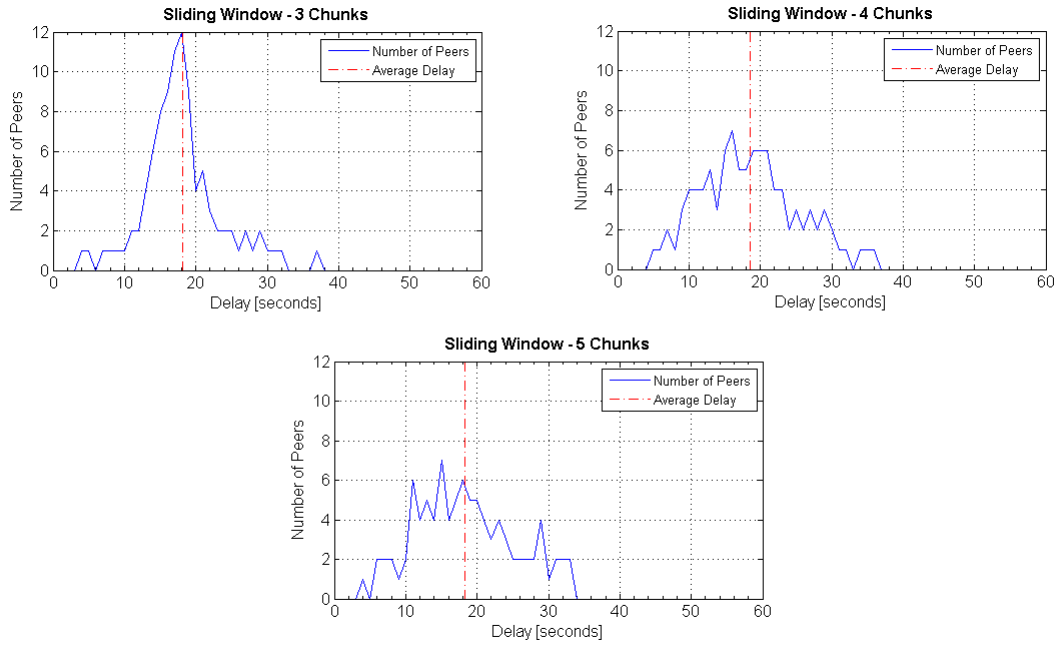

Figure 5.16: Histogram of delays between the creation of the chunk files in the server and the playback of the video stream on the receiving peers using an initial choke/unchoke interval of 5 seconds.

Figure 5.16 presents the histograms of the delays verified between the availability of the chunk files in the server and the reproduction of the stream on the receiving peers. The best results of this variable were achieved for the sliding window sizes of 3 and 5 chunks, with a delay of nearly 18 seconds, while for the sliding window sizes of 4 chunks achieved a delay of nearly 19 seconds.

Comparing with the results obtained in section 5.4, on the scenario with a sliding window size of 3 chunks, the delay decreased from 25 to 18 seconds.

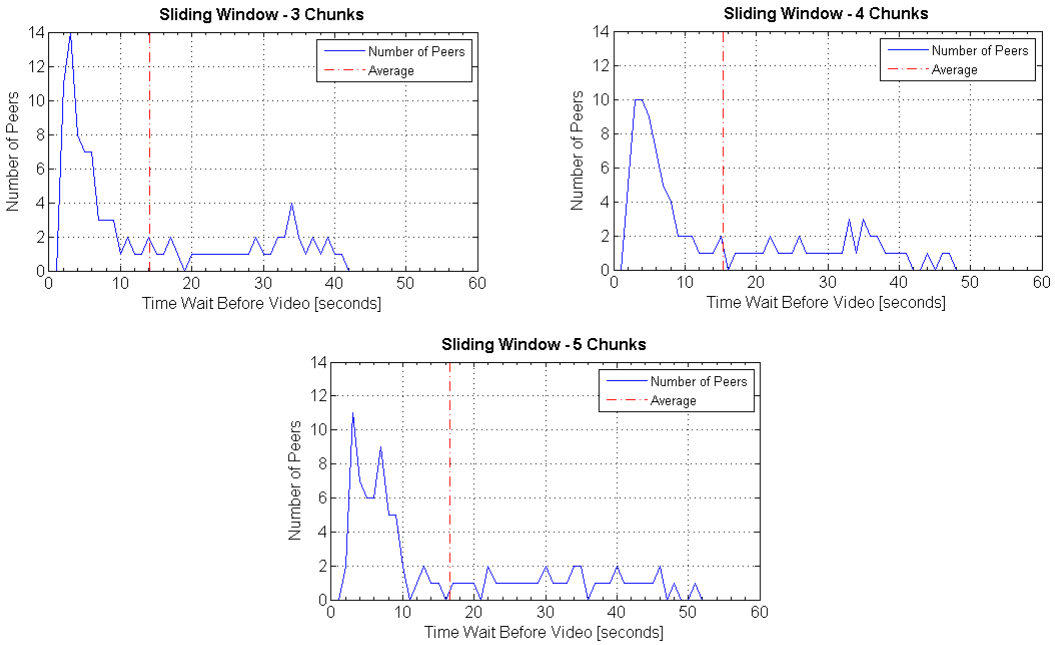

Figure 5.17: Histogram of elapsed time between stream request and playback start using an initial choke/unchoke interval of 5 seconds.

Figure 5.17 presents the histograms of the time elapsed until the content reproduction. The best result of this variable was achieved for the sliding window of 3 chunks, with a delay of 14 seconds; while for the sliding window of 4 and 5 chunks were obtained values of 15 and 17 seconds respectively.

Comparing these results with the ones obtained in section 5.4, when using a sliding window size of 3 chunks, the delay decreased almost by half, from 26 to 14 seconds.

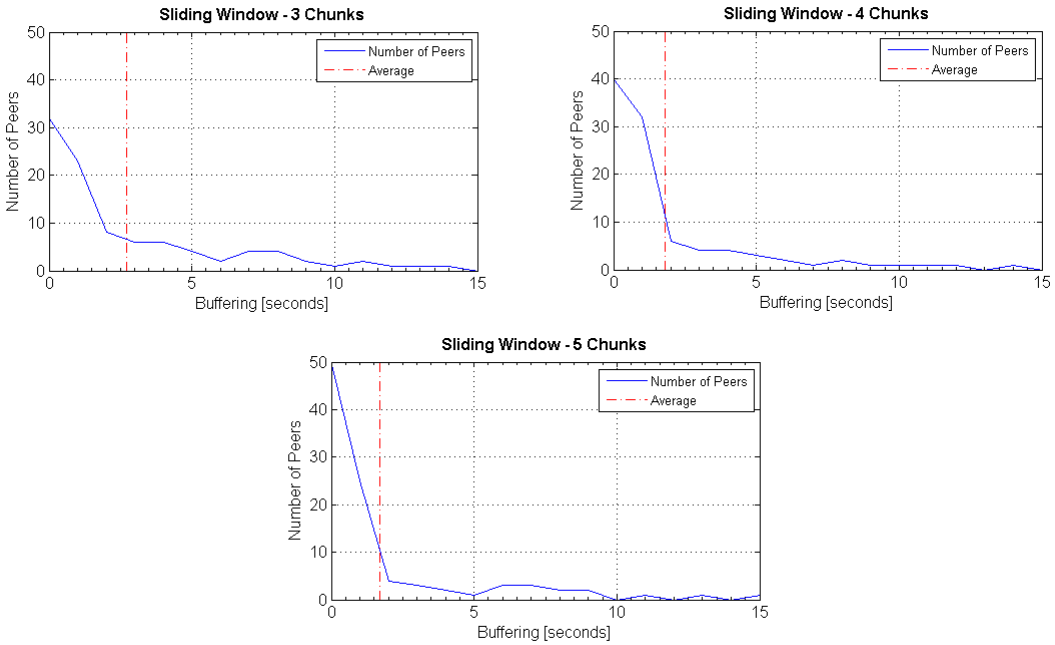

Figure 5.18: Histogram of delays caused by re-buffering mechanisms using an initial choke/unchoke interval of 5 seconds.

Figure 5.18 presents the histograms of delays caused by the pause-and-resume playback of video, while using an initial choke/unchoke interval of 5 seconds. The delay values achieved for the sliding window sizes of 3 and 4 chunks were 2.7 and 1.8 seconds respectively, while the best result was achieved for the sliding window size of 5 chunks, with a delay of 1.7 seconds.

When comparing these values with the ones of previous tests an increment in delays was verified from the 0.6 to 1.7 seconds, which indicates that these values should be carefully adjusted as peers have a lower probability of obtaining chunks in time.

5.5.3. Combining Number of Downloading Peers and Initial Choke/Unchoke Interval

The improvements verified by either varying the number of downloading peers or the initial choke/unchoke interval, in sections 5.5.1 and 5.5.2, were very promising due to their impacts in the obtained results, in which can be highlighted by the reception of the base layer by nearly 100% of the peers and the decrease of 50% on the elapsed time measured between the creation of the chunk files in the server and the playback of the video stream, and delay between entering in the swarm and the start of the video playback.

Given the similarities in the obtained results, it became clear the need to combine the variations of both the variables, number of downloading peers and initial choke/unchoke interval, in a new group of tests. In the following we will describe the results of these tests, which have maintained the same networks conditions as in previous tests, combining the variation of the number of downloading peers for 4, 6 and 8 peers, with the variation of the initial choke/unchoke interval from 1 to 10 seconds in steps of 1 second, using different sliding window sizes of 3, 4 and 5 chunks.

In Figures 5.19, 5.20 and 5.21 is shown the average value of the number of layers received by the peers in the swarm, respectively for 4, 6 and 8 downloading peers.

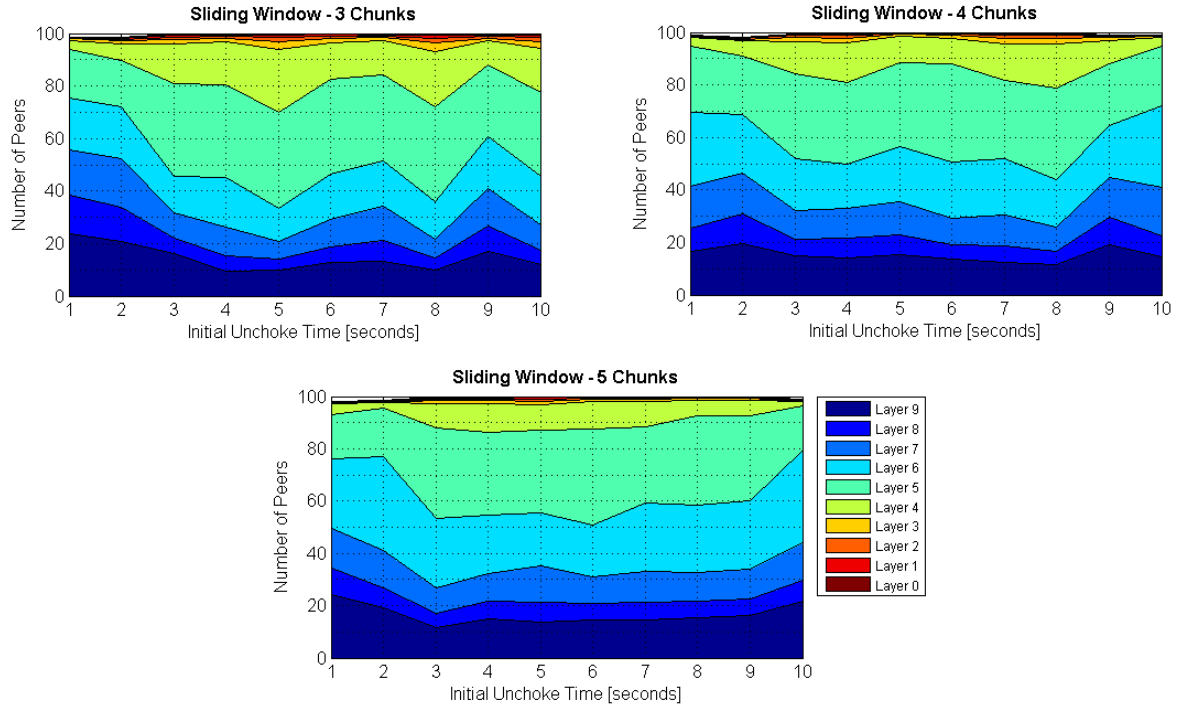

Figure 5.19: Average transmission layers received using 4 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

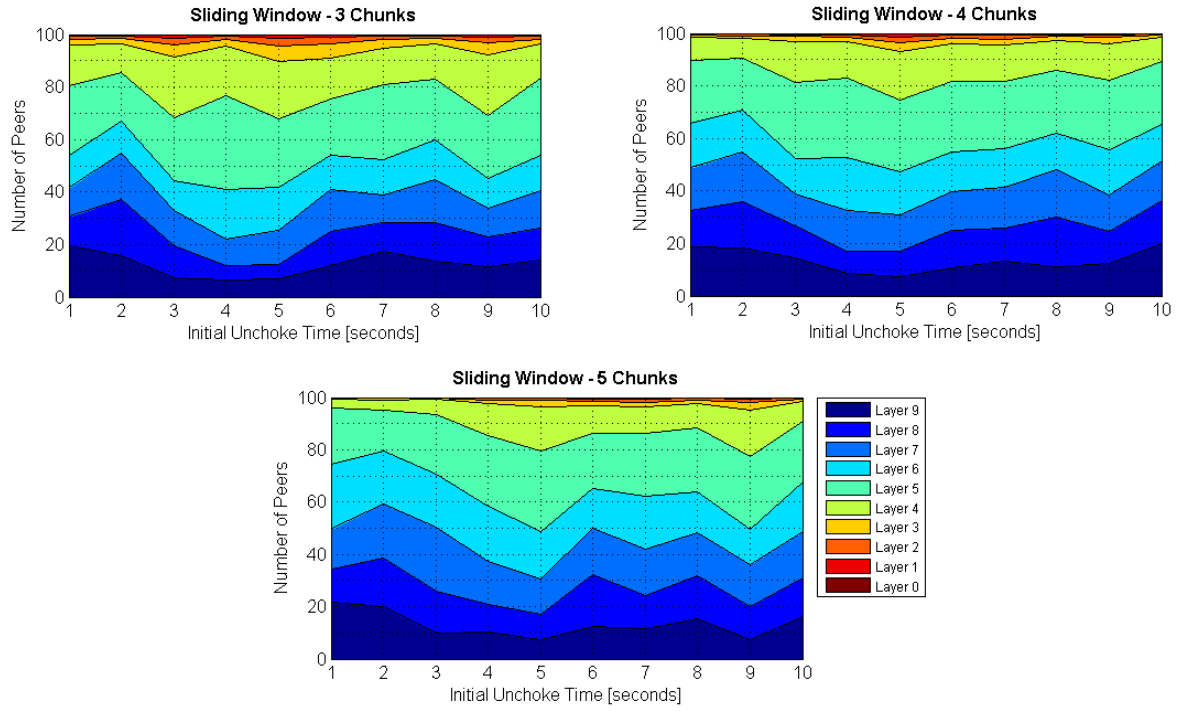

Figure 5.20: Average transmission layers received using 6 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

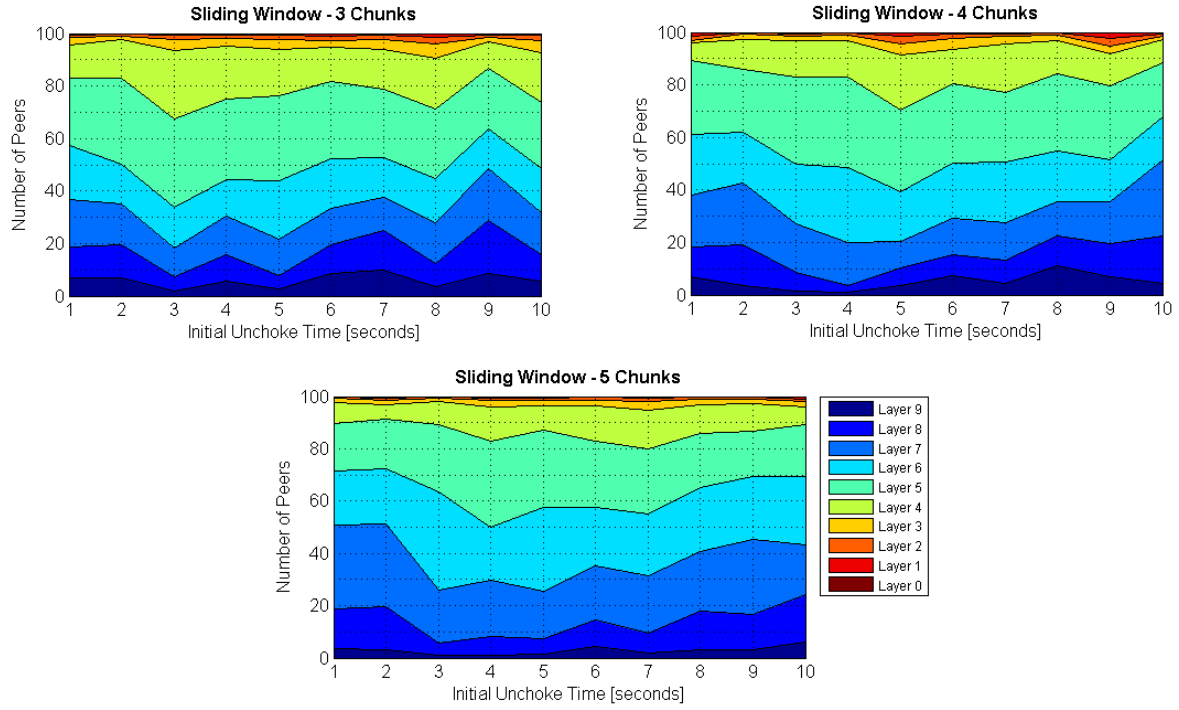

Figure 5.21: Average transmission layers received using 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

In Tables 5.5, 5.6 and 5.7 the values obtained in the tests are shown for the transmitted base (i.e. Layer 0), intermediate (i.e. Layer 5) and highest (i.e. Layer 9) layers.

Sliding Window Size – 3 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	98.6%	98.4%	99.7%	99.7%	99.9%	99.9%	99.8%	99.9%	99.4%	99.6%
Layer 5	93.8%	89.7%	81.1%	80.7%	69.9%	82.5%	84.2%	71.9%	87.8%	77.4%
Layer 9	23.8%	20.7%	16.3%	9.7%	10.1%	12.9%	13.2%	10.0%	17.0%	11.8%

Sliding Window Size – 4 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	99.1%	98.3%	99.5%	99.7%	99.9%	99.9%	99.7%	99.8%	99.2%	99.1%
Layer 5	94.9%	91.0%	84.1%	80.8%	88.5%	87.9%	81.8%	78.9%	87.8%	94.6%
Layer 9	16.6%	19.6%	14.7%	14.1%	15.5%	13.7%	12.5%	11.6%	19.2%	14.7%

Sliding Window Size – 5 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	98.1%	98.4%	99.6%	99.7%	99.9%	99.9%	99.8%	99.8%	99.7%	98.7%
Layer 5	93.1%	95.5%	88.1%	86.4%	87.2%	87.5%	88.5%	92.7%	92.5%	96.4%
Layer 9	24.3%	19.4%	11.7%	14.9%	13.6%	14.6%	14.4%	15.5%	16.2%	21.5%

Table 5.5: Average ratio of transmission layers received using 4 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Sliding Window Size – 3 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.00%	100.0%	100.00%	100.00%	100.0%	100.0%	100.0%
Layer 5	80.5%	85.7%	68.3%	76.8%	68.0%	75.3%	80.9%	83.0%	69.2%	83.3%
Layer 9	19.8%	15.7%	7.5%	6.4%	7.0%	12.0%	17.6%	13.8%	11.6%	14.0%

Sliding Window Size – 4 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	99.9%	100.0%	100.00%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	89.5%	90.6%	81.1%	82.9%	74.6%	81.6%	81.7%	85.9%	82.0%	89.3%
Layer 9	18.5%	18.2%	14.4%	8.5%	7.5%	10.8%	13.4%	11.1%	12.5%	19.8%

Sliding Window Size – 5 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	99.9%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	96.0%	95.0%	93.6%	85.6%	79.5%	86.3%	86.3%	88.4%	77.3%	90.9%
Layer 9	21.7%	20.2%	10.0%	10.2%	7.4%	12.6%	11.8%	15.3%	7.5%	16.4%

Table 5.6: Average ratio of transmission layers received using 6 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Sliding Window Size – 3 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	83.1%	83.2%	67.6%	75.2%	76.3%	81.6%	78.7%	71.2%	86.6%	73.7%
Layer 9	7.0%	6.9%	2.0%	5.5%	2.6%	8.7%	9.8%	3.4%	8.6%	5.6%

Sliding Window Size – 4 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.00%	100.00%	100.0%
Layer 5	89.5%	85.8%	82.9%	83.1%	70.4%	80.5%	77.2%	84.2%	79.4%	88.5%
Layer 9	7.2%	3.5%	1.6%	1.1%	3.6%	7.4%	4.6%	11.2%	6.8%	4.5%

Sliding Window Size – 5 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	89.6%	91.2%	89.1%	82.8%	87.3%	82.7%	80.1%	86.0%	86.8%	89.2%
Layer 9	3.7%	3.0%	1.1%	1.2%	1.5%	4.6%	1.7%	3.3%	3.1%	5.9%

Table 5.7: Average ratio of transmission layers received using 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

When comparing the Figures 5.19, 5.20, 5.21 and Tables 5.5, 5.6, 5.7, it can be verified that the solution with 4 downloading peers while presenting a lower probability of receiving the base layer guaranties a higher number of peers are able to get higher layers. It can be also verified the downward trend of peers receiving higher layers when increasing the number of downloading peers.

The advantage of the solutions with 6 and 8 downloading peers is that it is able of assuring that the base layer is received by every peer in the swarm.

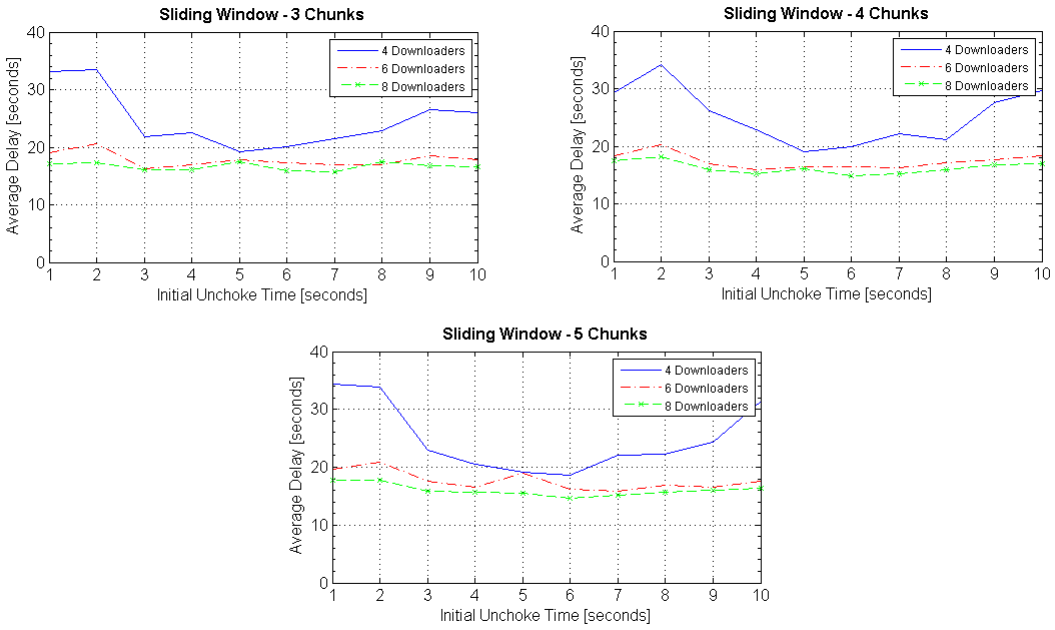

Figure 5.22: Histogram of average time values elapsed between the creation of the chunk files in the server and the playback of the video stream, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.22 presents the histograms of the average delays verified between the creation of the chunk files in the server and the playback of the stream on the receiving peers, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks. These figures show that the increment of the number of downloading peers to 6 and 8 significantly reduce these delays. The best results of this variable were achieved in the scenarios using 8 downloader peers, an initial choke/unchoke time of 6 seconds and sliding windows sizes of 4 and 5 chunks, respectively yielding average delays of 14.9 and 14.7 seconds.

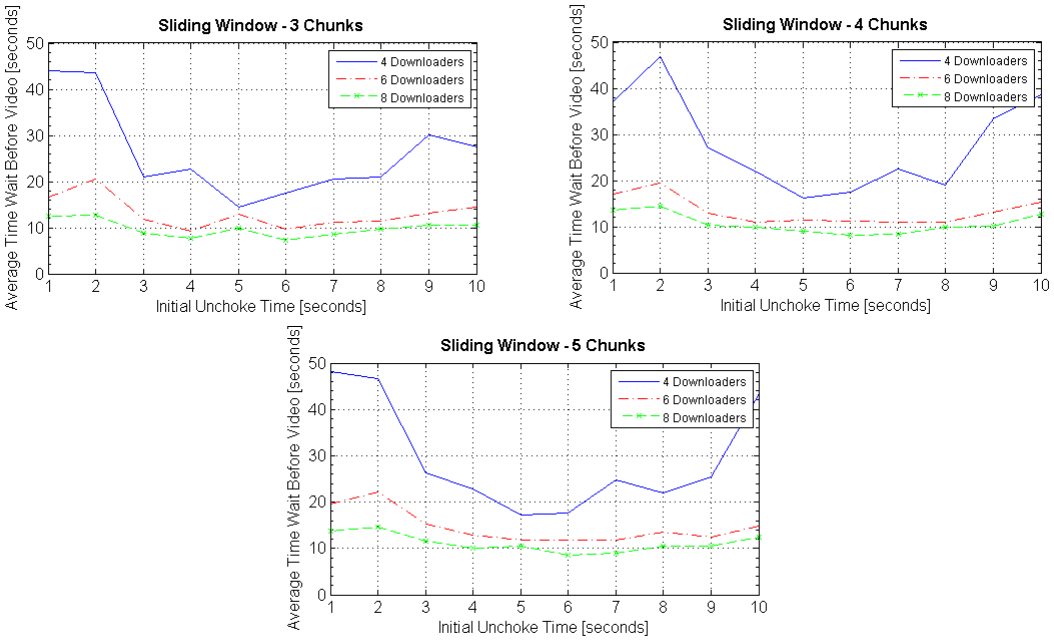

Figure 5.23: Histogram of average delays until the video playback, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.23 presents the histogram of the average elapsed time until the video playback, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks. These figures show that the increment of the number of downloading peers to 6 and 8 significantly reduce these delays. The best results of this variable were achieved in the scenarios using 8 downloader peers with sliding windows of 3 chunks and an initial choke/unchoke times of 4 and 6 seconds, yielding an average delay of 7.7 and 7.3 seconds.

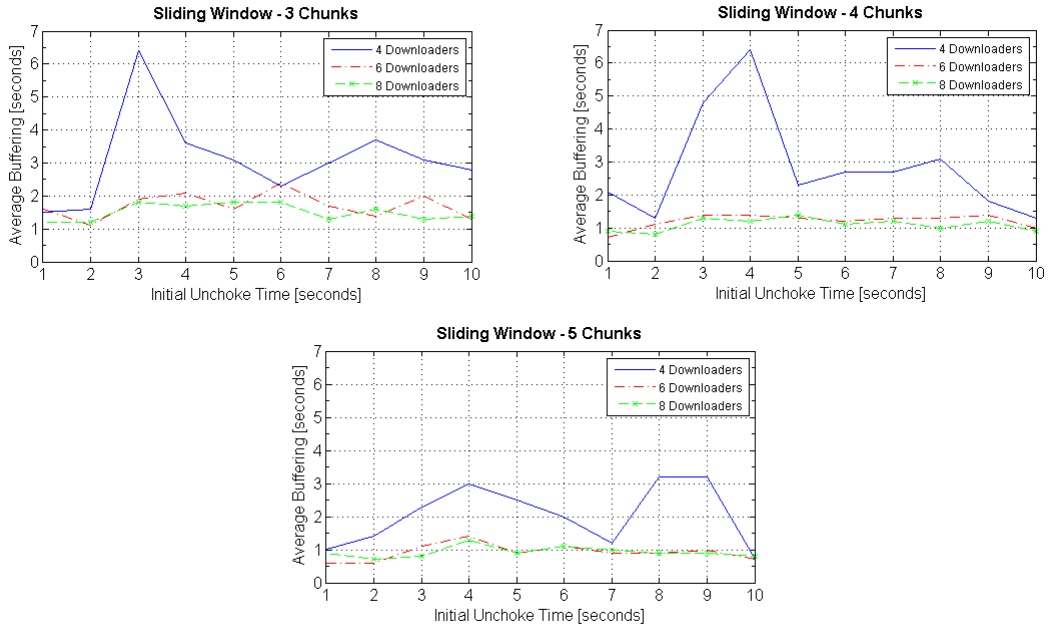

Figure 5.24: Histogram of the average delays caused by the re-buffering mechanisms, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.24 presents the histograms of the average delay caused by the buffering mechanisms, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks. These figures show that the increment of the number of downloading peers to 6 and 8 significantly reduce these delays. The best results were achieved in the scenarios using 6 downloader peers with sliding windows of 5 chunks and an initial choke/unchoke times of 1 and 2 seconds, causing an average delay of 0.6 seconds.

5.5.4. Summary

The results obtained show that by increasing of the number of downloader peers an increment of peers being able to receive at least the base layer was verified and an expressive reduction of the different delays. Regarding the adjustment of the initial choke/unchoke time; we verify a significant reduction in the average time values elapsed between the creation of the chunk files in the server and the playback of the video stream and in the delay of startup in video playback, while an increment in the delay caused by re-buffering was verified.

The results of combining both the number of downloading peers and the initial choke/unchoke time are able to achieve a significant reduction of the delays.

For the elapsed time verified between the original bit streams and content reproduction for each receiving peer, a result of nearly 14.7 seconds is obtained, while delay between content request and reproduction achieves 7.3 seconds. Regarding the delay caused by buffering mechanisms a value of 0.6 was reached.

These results demonstrated that besides the enhancements verified on the number of peers receiving more transmission layers and the very significant reduction of the peers not receiving any layer, some improvements are still required to reduce the value of the several delays measured and assure a high number of decoding of layers.

5.6. Base Layer Assurance Mechanism

The results obtained in the previous section continue far from the expected values, when considering a real-time distribution of H.264/SVC based on BitTorrent.

According to the results obtain in section 5.5, the time elapsed between the creation of the chunk files in the server and their playback on the receiving peers and especially the delay between content request and playback continue to be too high to be acceptable when compared with a conventional TV distribution system.

After a detailed analysis of the log files produced by the simulation framework in the tests performed for the previous sections of this chapter, the origin of those issues seems to be related with the time spent waiting for the base layer.

Given these issues, in the following a new base layer assurance mechanism is considered that was based in the method used in BitTorrent to request pieces. A new queue of requests was created with the aim to only attend requests for the base layer pieces, so a peer receiving the requests can attend to base layer chunks requests independently from the typical downloading peers that uses the choke/unchoke mechanism. This new mechanism enables a peer to download files from a neighbor peer besides its choke/unchoke connection state, so if a request arrives to a peer and if still there is available bandwidth to attend the request, the piece file is sent to the requesting peer.

This section of the study describes the tests performed with the simulation model previously used in sections 5.4 and 5.5, in order to analyze the effects of the mechanism capable of ensuring the reception of the base transmission layer on the receiving peers.

The tests performed considered the same H.264/SVC encoded sequence used in the previous sections (structure and characteristics previously described in section 4.1) and the same BitTorrent

environment, with a swarm of 100 peers and transmitting a bit stream during 900 seconds. The network capabilities of the peers were also the same as previously used and all peers entered the swarm in a random distribution along the first 30 seconds of the experiment.

Each test was repeated 5 times, combining a variation of the number of downloading peers of 4, 6 and 8 nodes, with a variation of the initial choke/unchoke interval from 1 to 10 seconds in steps of 1 second, with different sliding window sizes of 3, 4 and 5 chunks.

5.6.1. Base Layer Assurance Mechanism

Figures 5.25, 5.26 and 5.27 present the average values of the number of layers received by the peers in the swarm using a base layer assurance mechanism. In Tables 5.8, 5.9 and 5.10 the values for the transmitted base (i.e. Layer 0), intermediate (i.e. Layer 5) and highest (i.e. Layer 9) layers are presented.

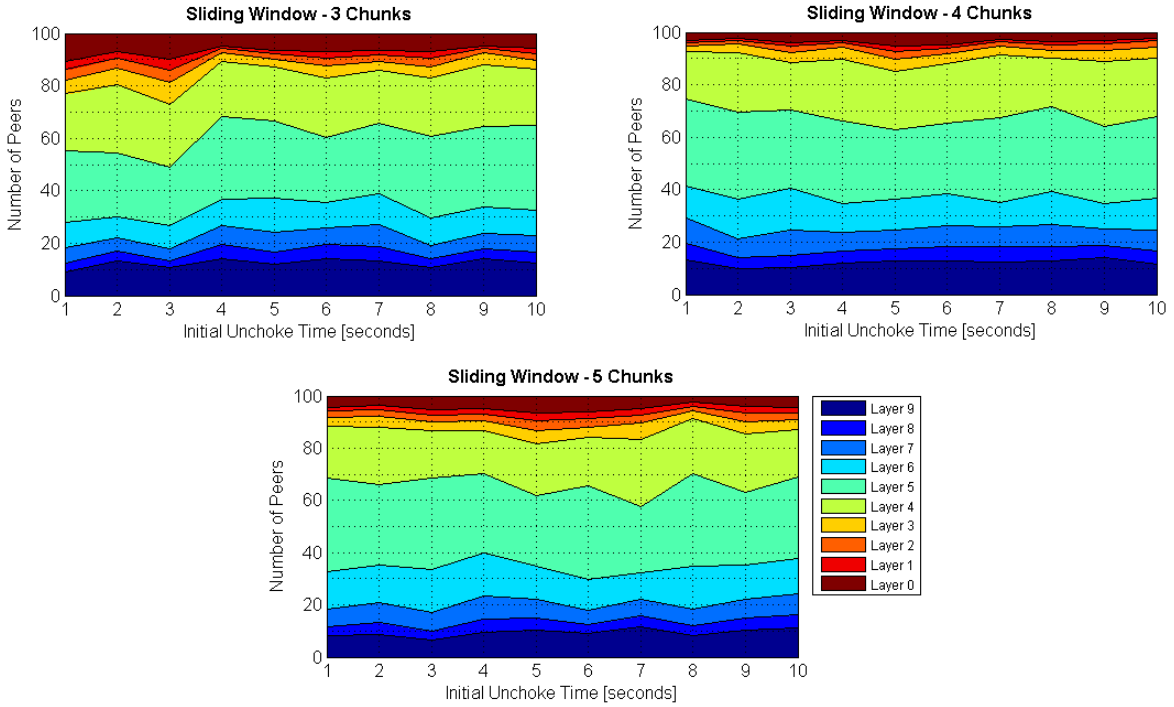

Figure 5.25: Average transmission layers received using base layer assurance mechanisms with 4 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

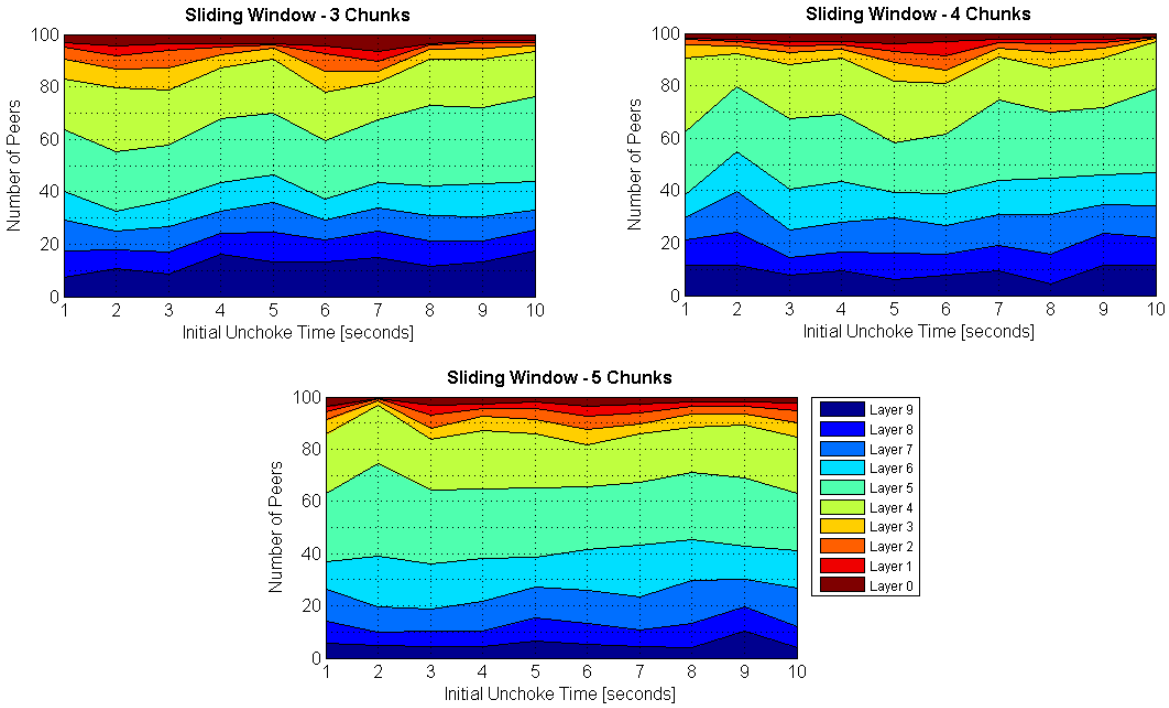

Figure 5.26: Average transmission layers received using base layer assurance mechanisms with 6 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

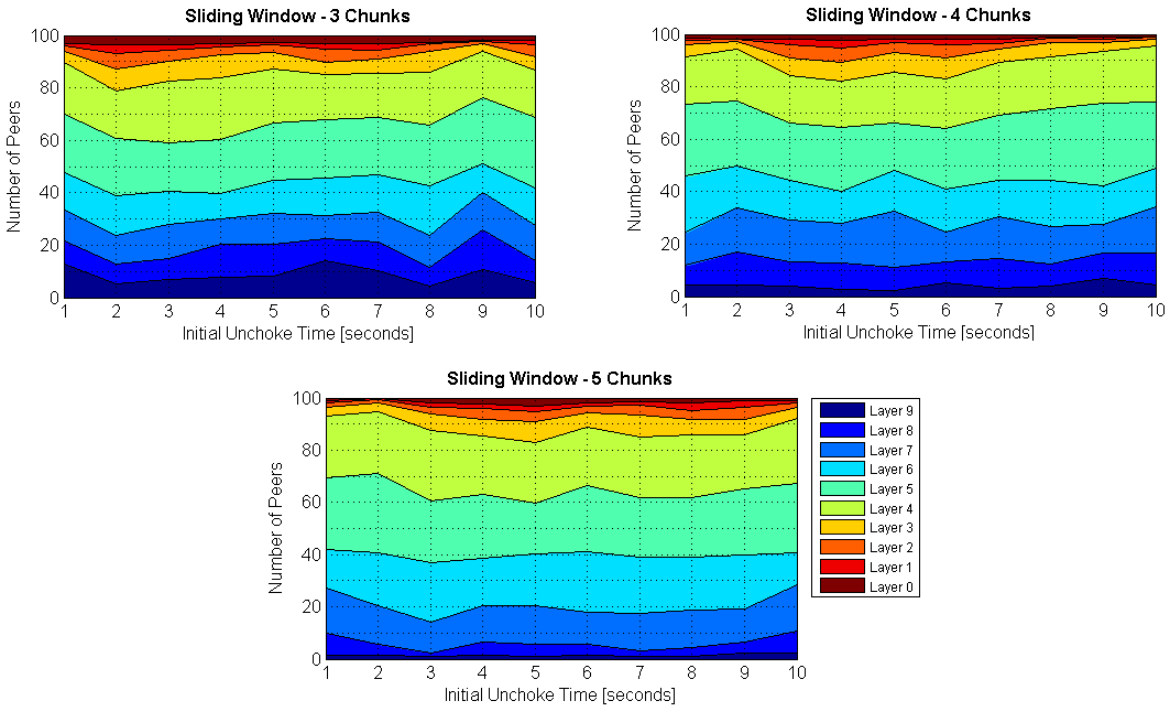

Figure 5.27: Average transmission layers received using base layer assurance mechanisms with 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Sliding Window Size – 3 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	55.3%	54.6%	49.0%	68.4%	66.7%	60.4%	65.9%	60.9%	64.7%	65.1%
Layer 9	9.0%	13.1%	10.5%	13.9%	11.9%	14.1%	13.0%	10.5%	13.9%	12.6%

Sliding Window Size – 4 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	74.5%	69.7%	70.4%	66.2%	62.7%	65.3%	67.6%	71.7%	64.3%	67.7%
Layer 9	13.1%	9.8%	10.1%	11.8%	12.7%	13.0%	12.6%	12.8%	13.9%	11.4%

Sliding Window Size – 5 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	68.6%	65.9%	68.8%	70.3%	62.0%	65.5%	57.6%	70.1%	63.3%	68.9%
Layer 9	8.1%	8.8%	6.5%	9.4%	10.4%	9.1%	11.5%	8.2%	10.2%	11.2%

Table 5.8: Average ratio of transmission layers received using base layer assurance mechanisms with 4 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Sliding Window Size – 3 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	63.8%	55.3%	57.6%	67.7%	69.9%	59.3%	67.3%	72.9%	72.0%	76.2%
Layer 9	7.2%	10.8%	8.6%	16.2%	13.2%	13.3%	14.9%	11.4%	13.4%	17.4%

Sliding Window Size – 4 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	62.2%	79.8%	67.6%	69.1%	58.2%	61.4%	74.4%	69.8%	71.6%	79.0%
Layer 9	11.5%	11.8%	8.0%	9.5%	6.2%	7.6%	9.6%	4.4%	11.5%	11.7%

Sliding Window Size – 5 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	63.0%	74.3%	64.2%	64.7%	65.0%	65.7%	67.5%	71.3%	69.1%	63.2%
Layer 9	5.6%	4.9%	4.5%	4.6%	6.7%	5.2%	4.4%	4.1%	10.2%	4.2%

Table 5.9: Average ratio of transmission layers received using base layer assurance mechanisms with 6 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Sliding Window Size – 3 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	70.2%	60.6%	59.2%	60.3%	66.6%	67.8%	68.6%	65.8%	76.1%	68.6%
Layer 9	12.7%	5.4%	6.8%	8.0%	8.4%	14.0%	10.4%	4.6%	10.8%	5.6%

Sliding Window Size – 4 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	73.4%	74.8%	66.1%	64.3%	66.0%	64.2%	69.3%	71.6%	73.9%	74.2%
Layer 9	4.2%	4.5%	3.8%	2.7%	2.2%	5.2%	3.0%	3.9%	6.9%	4.6%

Sliding Window Size – 5 Chunks										
	Initial Choke/Unchoke Interval [seconds]									
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	69.5%	71.0%	60.4%	63.3%	59.8%	66.5%	61.7%	61.8%	65.4%	67.5%
Layer 9	1.6%	1.3%	1.1%	1.6%	1.2%	1.5%	1.1%	1.1%	2.3%	2.2%

Table 5.10: Average ratio of transmission layers received using base layer assurance mechanisms with 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Comparing Figures 5.25, 5.26, 5.27 and the equivalent results on the Tables 5.8, 5.9, 5.10, it can be verified that the assurance of the base layer reception by all peers assures that all peers are able to

receive at least the base layer for any initial choke/unchoke interval or sliding window size. However it also shows a reduction on the reception of the intermediate and higher layers.

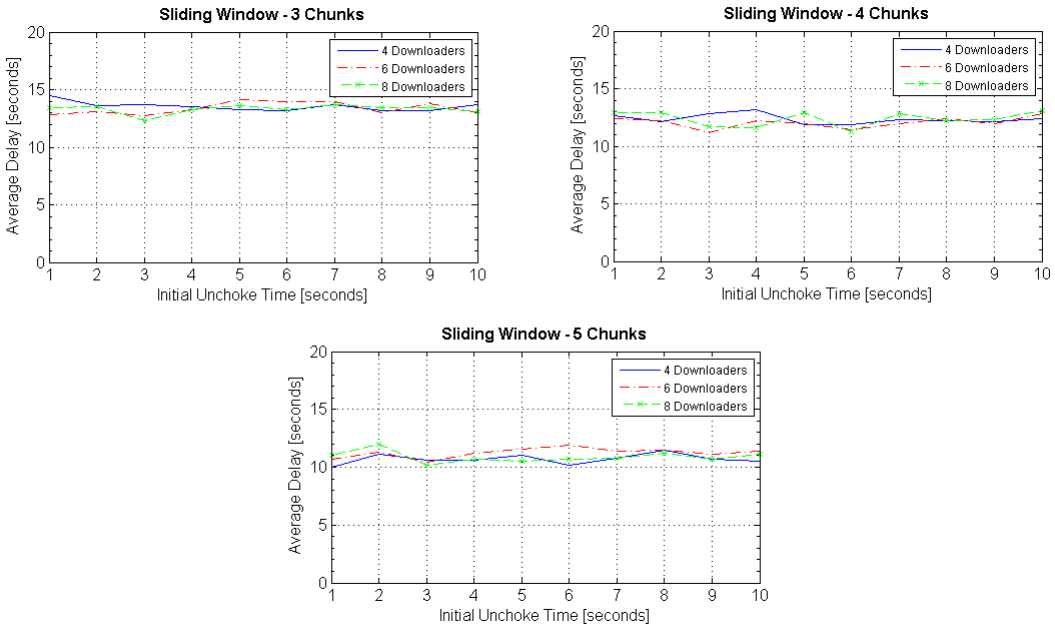

Figure 5.28: Histogram of average delays verified between the creation of the chunk files in the server and the playback of the video stream using base layer assurance mechanisms, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.28 presents the histograms of the time elapsed between the creation of the chunk files in the server and the playback of the stream on the receiving peers using a base layer assurance mechanism, for a combined variation of the number of downloading peers of 4, 6 and 8 with a variation of the initial choke/unchoke interval from 1 to 10 seconds in steps of 1 second, with the different sliding window sizes of 3, 4 and 5 chunks. The best values were achieved using a sliding window size of 5 chunks yielding an average delay of 10.2 seconds. In this case, the contribution of the choke/unchoke times and number of downloaders has been neglectable.

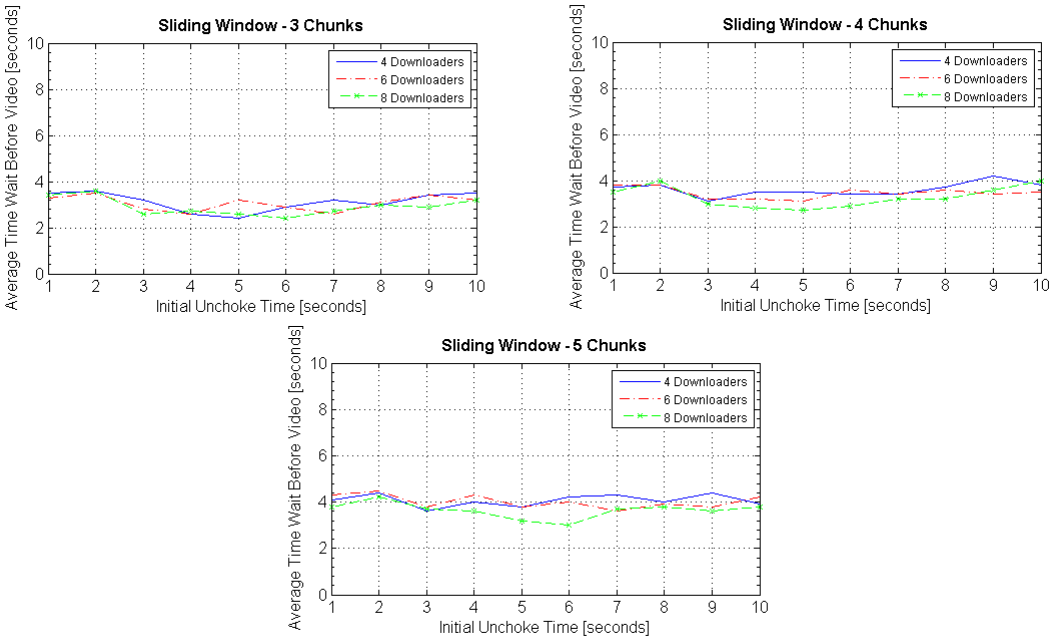

Figure 5.29: Histogram of average elapsed time until the video playback using base layer assurance mechanisms, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.29 presents the histogram of the average elapsed time until the video playback using base layer assurance mechanisms, for a combined variation of the number of downloading peers from 4, 6 and 8 with a variation of the initial choke/unchoke interval from 1 to 10 seconds in steps of 1 second, for the different sliding window sizes of 3, 4 and 5 chunks. The best values were achieved using a sliding window size of 3 chunks, 8 downloading peers and a choke/unchoke period of 6 seconds yielding an average delay of 2.4 seconds. Globally choke/unchoke periods of between 4 and 6 seconds tend to yield good results.

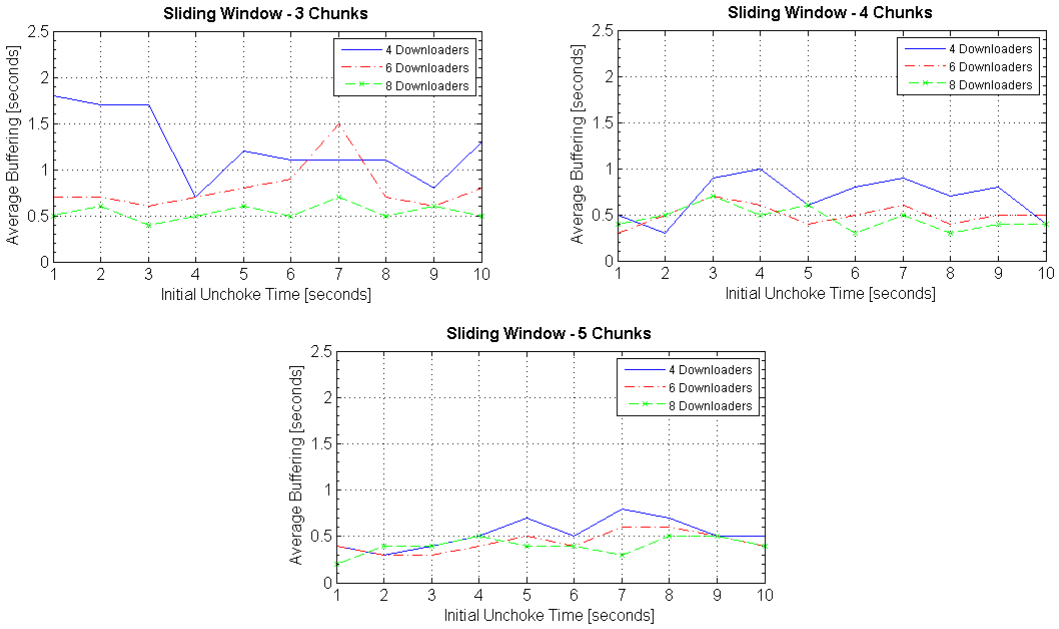

Figure 5.30: Histogram of the average delays caused by the re-buffering mechanisms using base layer assurance mechanisms, using 4, 6 and 8 downloading peers combined with the variation of the initial choke/unchoke interval for different sliding window sizes: 3, 4 and 5 chunks.

Figure 5.30 presents the histograms of the average delay caused by the pause-and-resume in playback using the base layer assurance mechanisms, for a combined variation of the number of downloading peers from 4, 6 and 8 with a variation of the initial choke/unchoke interval from 1 to 10 seconds in steps of 1 second, for the different sliding window sizes of 3, 4 and 5 chunks. The best results were achieved in the scenarios using 6 downloading peers with sliding windows of 5 chunks and an initial choke/unchoke times of 1 second, yielding an average delay of 0.2 seconds.

5.6.2. Summary

The results obtained have shown that the reception of the base layer in every peer is completely fulfilled, even with the cost of a small decrease in the number of peers receiving intermediate and higher layers.

Above all, the results have shown a significant reduction of the delays; for the time elapsed between the original bit streams and content reproduction in each receiving peer, a value of 10.2 seconds was obtained, while the delay between content request and playback reached a value of 2.4 seconds. Finally for the delay caused by re-buffering mechanisms has reached a value of 0.2 seconds.

These results demonstrated that a base layer assurance mechanism plays a very important role in the reduction of the several delays measured and that this mechanism could be the main enhancement to make those values acceptable in a real-time distribution of H.264/SVC using a BitTorrent overlay.

Given the results in the last sections for the best group of combined results of the analyzed variables, i.e., the number of transmission layers received by the peers, the delay verified between the creation of the chunk files in the server and the playback of the video stream, the elapsed time until the video playback and the delay caused by the re-buffering mechanisms, in the following sections, we have chosen to performed the tests using 8 downloading peers with a sliding window of 5 chunks and initial choke/unchoke intervals of 4, 5 and 6 seconds.

5.7. Impact of Peer Churn in the Swarm

The results obtained in section 5.6, are now closer to the objectives set in this study. The system is capable of distributing H.264/SVC encoded video sequences to every peer in a swarm with acceptable performance in terms of delays.

However, until now the distribution of H.264/SVC has been tested without considering peer churn, which might cause a severe degradation in the quality of the reception. While we relied in the reliability inherited from BitTorrent P2P technology, it also needs to be tested.

For this purpose, in this section several tests are performed focusing in the reliability of the swarm, making abrupt variances in format, in the number of participating peers and also measuring the impact of the entrance or abandon of peers during the transmission of the encoded video sequence.

The simulation model used in section 5.6 was also used in this section to simulate peer churn.

The peer churn was simulated using the random entrance and exiting of peers to and from the swarm. As previously all peers entered the swarm in a random distribution along the first 30 seconds of the experiment. The abandon of the swarm was performed in the last half of the stream, starting nearly 450 seconds after the setup of the distribution. The tests where repeated with different churn rates, from 0% to 100% with increments of 10%. Peers that performing churn were chosen randomly and each test was repeated 10 times, for each peer churn rate.

Given the results obtained in the previous section the tests were performed using 8 downloading peers, with a sliding window of 5 chunks and varying the initial choke/unchoke intervals for 4, 5 and 6 seconds.

The tests performed used the same H.264/SVC encoded sequence was used in the previous sections (structure and characteristics previously described in section 4.1) and the same BitTorrent environment, with a swarm of 100 peers and transmitting a bit stream during 900 seconds. The network capabilities of the peers were also the same as previously used.

5.7.1. Measuring the Impact of Peer Churn in the Swarm

Figures 5.31 shows the average value of the number of layers received by the peers in the swarm, during the whole period of 900 seconds, considering different peer churn ratios and using different initial choke/unchoke intervals of 4, 5 and 6 seconds. Table 5.11 present the values for the transmitted base (i.e. Layer 0), intermediate (i.e. Layer 5) and highest (i.e. Layer 9) layers.

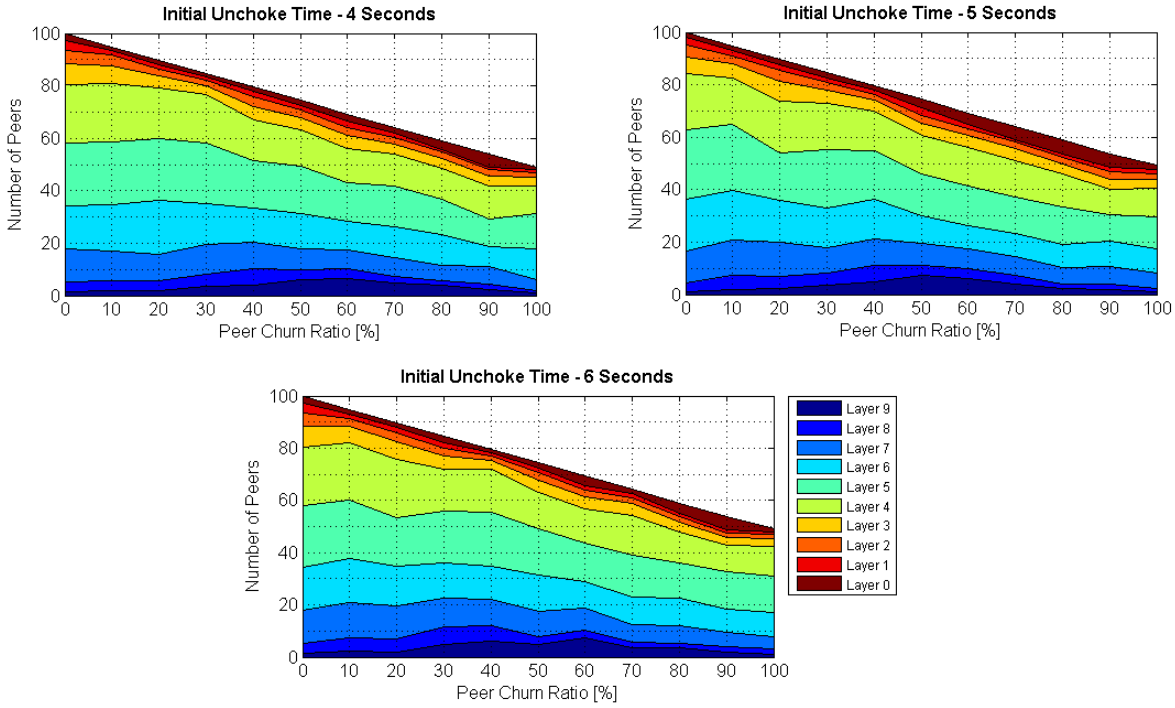

Figure 5.31: Average transmission layers received varying the peer churn ratio for different initial choke/unchoke intervals: 4, 5 and 6 seconds.

Initial Choke/Unchoke Interval – 4 Seconds											
	Peer Churn Ratio [%]										
	0	***10***	***20***	***30***	***40***	***50***	***60***	***70***	***80***	***90***	***100***
Layer 0	100.0%	94.9%	89.8%	84.6%	79.5%	74.4%	69.3%	64.2%	58.9%	53.9%	49.1%
Layer 5	58.1%	58.6%	59.9%	58.2%	51.6%	49.3%	43.2%	41.7%	36.8%	29.2%	31.1%
Layer 9	1.5%	1.9%	2.1%	3.5%	4.0%	6.2%	6.5%	5.0%	3.9%	2.2%	1.0%

Initial Choke/Unchoke Interval – 5 Seconds											
	Peer Churn Ratio [%]										
	0	***10***	***20***	***30***	***40***	***50***	***60***	***70***	***80***	***90***	***100***
Layer 0	100.0%	94.9%	89.8%	84.6%	79.5%	74.4%	69.3%	64.1%	58.9%	53.8%	49.2%
Layer 5	62.7%	65.0%	54.0%	55.1%	54.9%	46.1%	41.5%	37.3%	33.2%	30.5%	29.8%
Layer 9	1.1%	1.8%	2.3%	3.7%	5.0%	7.3%	6.3%	3.9%	2.4%	1.8%	1.1%

Initial Choke/Unchoke Interval – 6 Seconds											
	Peer Churn Ratio [%]										
	0	***10***	***20***	***30***	***40***	***50***	***60***	***70***	***80***	***90***	***100***
Layer 0	100.0%	94.9%	89.8%	84.6%	79.5%	74.4%	69.3%	64.2%	58.9%	53.7%	49.2%
Layer 5	58.1%	60.2%	53.3%	55.9%	55.6%	49.2%	43.5%	38.9%	36.1%	32.5%	31.1%
Layer 9	1.5%	2.1%	2.1%	5.0%	5.9%	4.7%	7.3%	3.8%	3.4%	1.8%	1.2%

Table 5.11: Average ratio of transmission layers received varying the peer churn ratio for different initial choke/unchoke intervals: 4, 5 and 6 seconds.

Figure 5.31 and the corresponding values in Table 5.11 show the evolution of the number of transmission layers received by the peers in the swarm as the chunks are being transmitted. It can be verified that the number of peers receiving the H.264/SVC encoded sequence, decreases linearly with the increment of the churn rate.

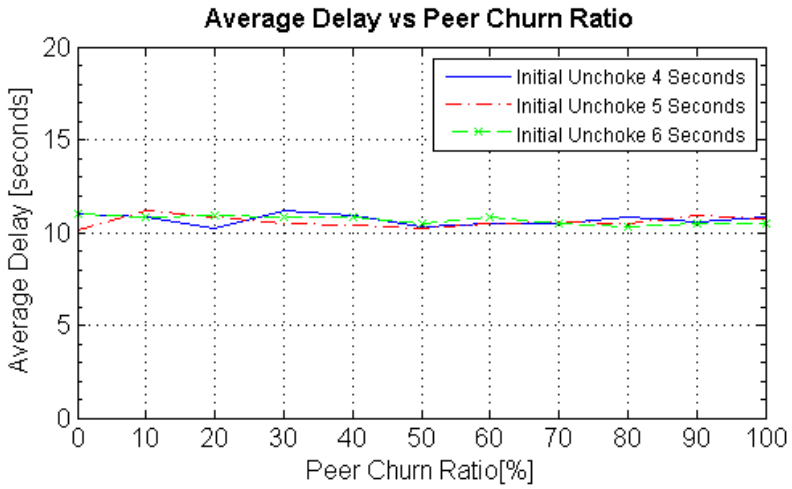

Figure 5.32: Histogram of average delays verified between the creation of the chunk files in the server and the playback of the video stream varying the peer churn ratio for different initial choke/unchoke intervals: 4, 5 and 6 seconds.

Figure 5.32 presents the histograms of the delays verified between the creation of the chunk files in the server and the playback of the stream on the receiving peers for the variation of the peer churn ratio, for different initial choke/unchoke intervals of 4, 5 and 6 seconds. The values present a constant behavior with small variations between 10 and 11 seconds, being consistent with the results obtained in Figure 5.28.

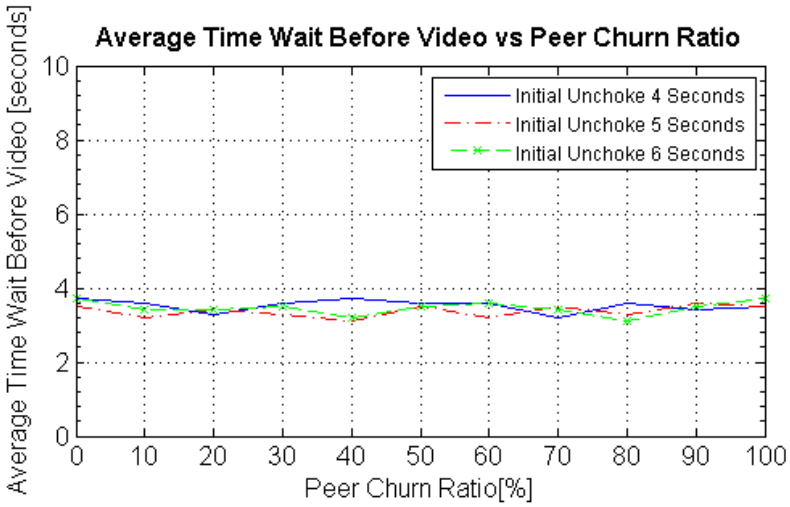

Figure 5.33: Histogram of average elapsed time until the video playback varying the peer churn ratio for different initial choke/unchoke intervals: 4, 5 and 6 seconds.

Figure 5.33 presents the histogram of the average elapsed time until the video playback for the variation of the peer churn ratio, for different initial choke/unchoke intervals of 4, 5 and 6 seconds. The values present a constant behavior with small variations between 3 and 4 seconds. These results are consistent with the ones obtained in Figure 5.29.

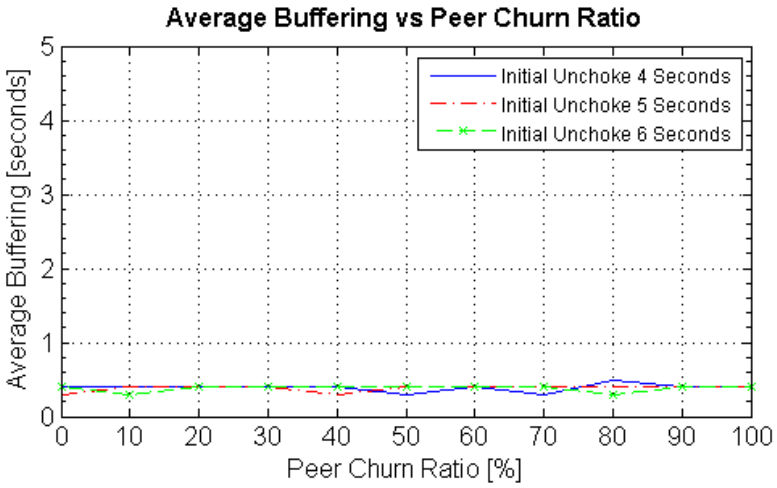

Figure 5.34: Histogram of the average delays caused by the re-buffering mechanisms varying the peer churn ratio for different initial choke/unchoke intervals: 4, 5 and 6 seconds.

Figure 5.34 presents the histograms of the average delay caused by the buffering mechanisms, for the variation of the peer churn ratio, for different initial choke/unchoke intervals of 4, 5 and 6 seconds. The obtained values present a constant behavior between 0.3 and 0.5 seconds.

5.7.2. Summary

The results obtained in these tests have shown that as the peer churn increases, the average number of received transmission layers decreases linearly. On average, the number of layers received is independent from the number of peers in the swarm.

It can also be verified that the delays maintain their values constant as peer churn increases, being consistent with the results obtain in section 5.6.

Comparing these values with the results obtained in section 5.6, we verify that for the delay between the generation of the original bit streams and content reproduction, in section 5.6 a value of 10.2 seconds was obtained and in this section the values fluctuate between 10 and 11 seconds; for the elapsed time until the playback on the receiving peers, in section 5.6 a value of 2.4 seconds was obtained and in this section the values fluctuate between 3 and 4 seconds; finally for the delay caused by buffering mechanisms, in section 5.6 a best value of 0.2 was reached, in this section the values vary between 0.3 and 0.5 seconds.

5.8. Alternative Piece Selection Method

In previous sections we were able to reduce the delays associated with H.264/SVC video distribution using a BitTorrent P2P overlay. However the effort of an in-time delivery of the lowest layers has come with the drawback of a reduction in the number of peers being able to receive higher layers, causing a reduction in scalability qualities given by SVC.

Particularly, the introduction of the base layer assurance mechanism, in section 5.6, has caused a significant reduction in the several delays, by giving a higher probability for the transmission of the base layer when compared with all the other layers.

Until this point we haven't considered any solution for data distribution among peers. In fact, chunks are delivered as requests arrive, which tends to result in a high probability of one peer serving several layers to another requesting peer. Ideally we could expect an increment of resilience when different chunks of distinct layers are spread to distinct peers, which in turn may still exchange them.

While in non-BitTorrent solutions like [19] a routing solution has been proposed for SVC that explores multipath for different layers targeting the implementation of a low-delay P2P streaming, no parallel solution has been investigated for BitTorrent.

Given these considerations, in this section we define a new piece selection method that lowers the probability of sending all layers to a certain peer. While in previous tests, when a peer receives a request for a chunk, if it is unchoked and the bandwidth still allows it, it would send the chunk piece file to the requesting peer, in the following we introduced a mechanism that, at the serving peer, will apply a random decision algorithm to decide if that chunk will be transmitted. The random probability that is applies depends on the layer of the chunk.

To achieve this, the following mechanism was implemented in the simulation model: given a request for a chunk, if the source peer is unchoked and while the bandwidth allows it, a random number is generated using an uniform distribution (between 0 and 1); given the number of the layer requested, the random number is compared with a pre-defined probability allocation function; if the value given by the probability allocation function for that layer is greater or equal to the generated random number, the chunk file piece is sent to the requesting peer, if it's lower, no answer is retrieved.

As distinct probability allocation functions could be used, in the following we considered twelve solutions, represented in Table 5.12.

	n	**Probability [P]**											
		1	***Steps***	***1/n***	***1/2n***	***1/2n'***	***1/3n***	***1/3n'***	***1/4n***	***1/4n'***	***1/5n***	***1/5n'***	***$1/n^2$***
Layer 0	1	1	1	1	1/2	1	1/2	1	1/4	1	1/5	1	1
Layer 1	2	1	1/2	1/2	1/4	1/4	1/6	1/6	1/8	1/8	1/10	1/10	1/4
Layer 2	3	1	1/2	1/3	1/6	1/6	1/9	1/9	1/12	1/12	1/15	1/15	1/9
Layer 3	4	1	1/2	1/4	1/8	1/8	1/12	1/12	1/16	1/16	1/20	1/20	1/16
Layer 4	5	1	1/4	1/5	1/10	1/10	1/15	1/15	1/20	1/20	1/25	1/25	1/25
Layer 5	6	1	1/4	1/6	1/12	1/12	1/18	1/18	1/24	1/24	1/30	1/30	1/36
Layer 6	7	1	1/4	1/7	1/14	1/14	1/21	1/21	1/28	1/28	1/35	1/35	1/49
Layer 7	8	1	1/8	1/8	1/16	1/16	1/24	1/24	1/32	1/32	1/40	1/40	1/64
Layer 8	9	1	1/8	1/9	1/18	1/18	1/27	1/27	1/36	1/36	1/45	1/45	1/81
Layer 9	10	1	1/8	1/10	1/20	1/20	1/30	1/30	1/40	1/40	1/50	1/50	1/100

Table 5.12: Values of probability of each transmission layer according to the probability schema used in the piece selection method.

The tests were performed using the same H.264/SVC encoded sequence of previous sections (structure and characteristics previously described in section 4.1) and the same BitTorrent environment, with a swarm of 100 peers and transmitting a bit stream during 900 seconds. The network capabilities of the peers were also the same as previously used and all peers entered the swarm in a random distribution along the first 30 seconds of the experiment.

The tests were made using 8 downloader peers with a sliding window of 5 chunks and an initial choke/unchoke timer of 5 seconds, each test was repeated 5 times.

5.8.1. Results of the Alternative Piece Selection Method

Figure 5.35 shows the average value of the number of layers received by the peers in the swarm according to the probability schema used in the piece selection method, as shown in Table 5.13.

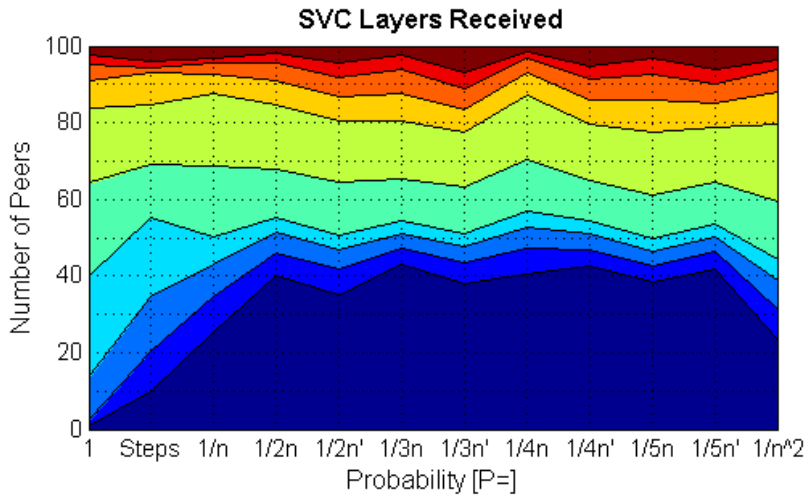

Figure 5.35: Average transmission layers received according to the probability schema used in the piece selection method.

	Probability [P]											
	1	***Steps***	***1/n***	***1/2n***	***1/2n'***	***1/3n***	***1/3n'***	***1/4n***	***1/4n'***	***1/5n***	***1/5n'***	***$1/n^2$***
Layer 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Layer 5	64.3%	68.9%	68.9%	67.9%	64.7%	65.2%	63.0%	70.3%	64.9%	61.1%	64.6%	59.7%
Layer 9	1.1%	9.9%	25.3%	40.1%	35.3%	43.0%	38.2%	40.5%	42.5%	38.5%	42.0%	23.1%

Table 5.13: Ratio of the average transmission layers received according to the probability schema used in the piece selection method.

When comparing Figure 5.35 with the corresponding values on Table 5.13, it can be verified that every solution can guarantee the reception of the base transmission layer to every peer in the swarm. It can be also verified that the solution using the probability schema '1/(3n)' guaranties that a higher number of peers are able to get the higher layer (Layer 9) and the solution using the probability schema '1/(4n)' guaranties a higher number of peers capable of receiving the intermediate layers (Layer 5).

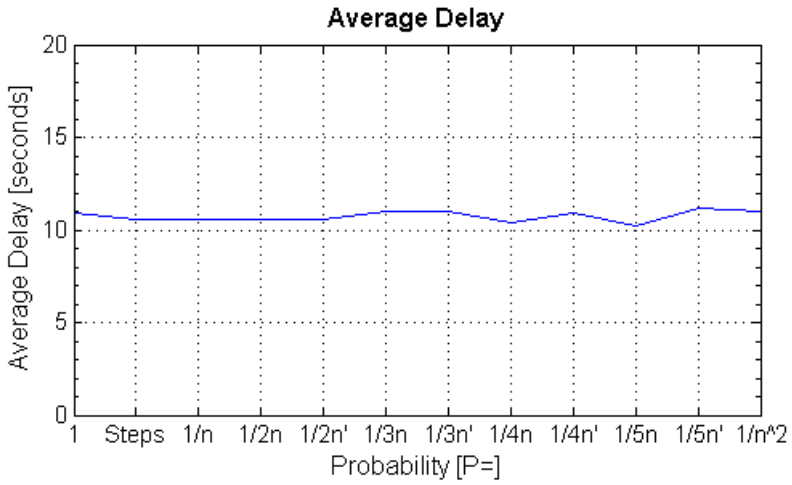

Figure 5.36: Histogram of the average delays verified between the availability of the chunk files in the server and their playback on the receiving peers according to the probability schema used in the piece selection method.

Figure 5.36 presents the histograms of the average time elapsed between the creation of the chunk files in the server and the playback of the stream on the receiving peers according to the probability schema used in the piece selection method. The values do not show a considerable difference between probability functions, with small variations between 10 and 11 seconds.

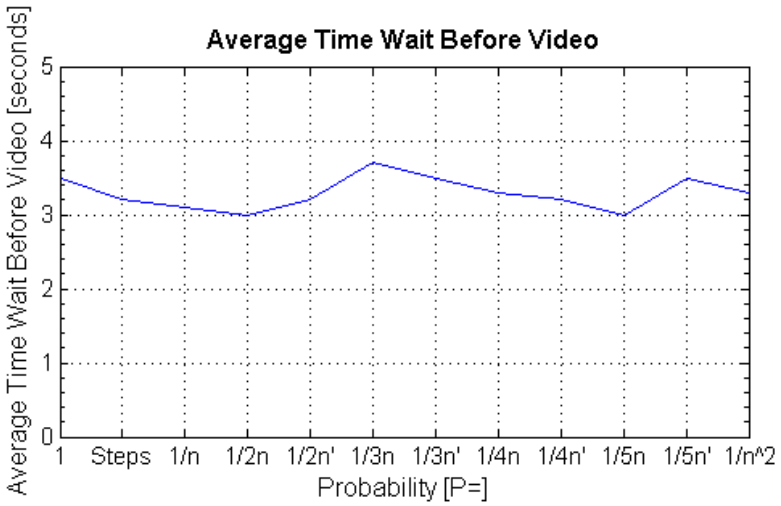

Figure 5.37: Histogram of the average elapsed time until playback on each receiving peer according to the probability schema used in the piece selection method.

Figure 5.37 presents the histogram of the delays between video request and playback according to the probability schema used in the piece selection method. The values present a nearly constant behavior with small variations between 3.0 and 3.7 seconds.

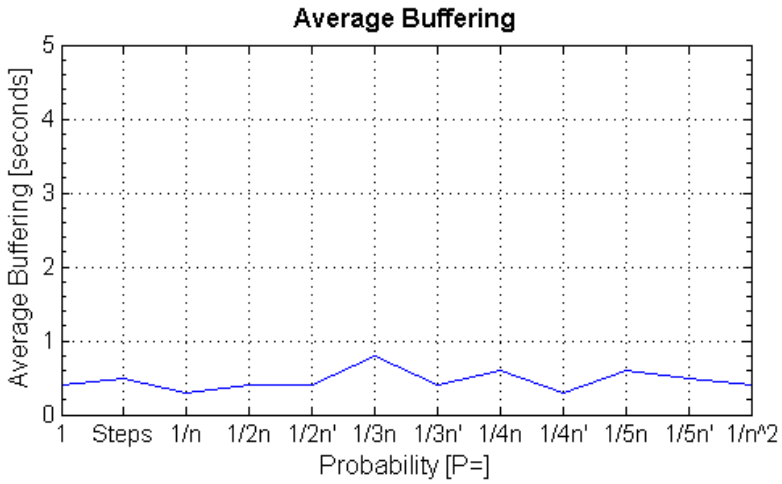

Figure 5.38: Histogram of the average delays verified by the re-buffering mechanisms for each receiving peer according to the probability schema used in the piece selection method

Figure 5.30 presents the histograms of the average delay caused by the re-buffering according to the probability schema used in the piece selection method. The values present a constant behavior with small variations between 0.3 and 0.8 seconds.

5.8.2. Summary

The results obtained showed that the introduction of the response of a piece request based on the probability of the transmission layer has a significant impact in the number of layers received by the peers in the swarm.

Comparing the value of peers receiving the higher layer (Layer 9) on Table 5.13 for a piece selection method using a probability schema of '1/3n', with the values for Table 5.10 in section 5.6 using 8 downloading peers with an initial choke/unchoke interval of 5 seconds and a sliding window of 5 chunks, it can be verified an extraordinary growth of nearly 3483%, from 1.2% to 43.0%.

In terms of delays, the introduction of the response of a piece request based on the probability of the transmission layer seems to not reproduce significant effects.

These results have demonstrated that the introduction of a probabilistic response to a piece request based on the probability of the transmission layer plays a very important role in the number of layers received by the peers.

5.9. Chapter Summary

In the previous sections of this chapter we described the implemented BitTorrent simulation framework, the performed tests methods and discussed the obtained results for each scenario.

In this section, the results obtained for each test method were compared and analyzed, in order to investigate the variations observed in main variables throught the tests in the previous sections of this chapter.

5.9.1. Number of Transmission Layers Received

Table 5.14 presents the number of transmission layers received by the peers in the swarm for the best test scenarios of each method, only the values for the transmitted base (i.e. Layer 0), intermediate (i.e. Layer 5) and highest (i.e. Layer 9) layers are presented.

Section – Test Method	**Best Test Scenario**	**Transmission Layers Received [%]**		
		Layer 0	***Layer 5***	***Layer 9***
Section 5.3 - Sliding Window Piece Selection Method	Sliding Window - 5 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	89.6%	85.3%	7.5%
Section 5.4 - Video Buffering Techniques	Sliding Window - 5 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	98.7%	96.4%	21.5%
Section 5.5 - Optimization of the Number of Downloading Peers and the Initial Choke/Unchoke Interval	Sliding Window - 5 Chunks 6 Downloading Peers Initial Choke/Unchoke Interval - 1 sec	100%	96.0%	21.7%
Section 5.6 - Base Layer Assurance Mechanism	Sliding Window - 4 Chunks 6 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	100%	76.2%	17.4%
Section 5.8 - Alternative Piece Selection Method	Sliding Window - 5 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval - 5 sec Probability Schema – 1/4n	100%	70.3%	40.5%

Table 5.14: Number of transmission layers received by the peers in the swarm for the best test scenarios of each method.

Comparing the obtained results in Table 5.14, it can be verified that the video buffering techniques introduced in the section 5.3 combined with the increase of the number of downloading peers, are the main responsibles for assuring that all peers are able to receive at least the base layer. However for the intermediate and higher layers, the increment of the number of downloading peers and the decrease of the initial choke/unchoke interval appears to have a more significant contribution in the number of received layers. Finally, the introduction of an alternative piece selection method has demonstrated a significant contribution in the number of the higher layers received in lieu to the intermediate layers.

5.9.2. Delay Verified Between the Availability of Chunk Files and Playback

In Table 5.15, the delay verified between the availability of the chunks files and the video playback for the best test scenarios of each method is presented.

Section – Test Method	Best Test Scenario	Delay Verified Between the Availability of Chunk Files and Playback [sec]
Section 5.3 - Sliding Window Piece Selection Method	Sliding Window - 3 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	15 sec
Section 5.4 - Video Buffering Techniques	Sliding Window - 3 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	25 sec
Section 5.5 - Optimization of the Number of Downloading Peers and the Initial Choke/Unchoke Interval	Sliding Window – 4, 5 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval - 6 sec	6.0 sec
Section 5.6 - Base Layer Assurance Mechanism	Sliding Window - 5 Chunks 4, 6, 8 Downloading Peers Initial Choke/Unchoke Interval – N/A	10.2 sec
Section 5.8 - Alternative Piece Selection Method	Sliding Window - 5 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval - 5 sec Probability Schema – Any	10 sec < Delay < 11 sec

Table 5.15: Delay verified between the availability of the chunk files and playback of the video for the best test scenarios of each method.

The results in Table 5.14, show that the increase of the number of downloading peers and the decrease of the initial choke/unchoke interval appears to have a significant contribution to reduce the delay verified between the availability of the chunk files and playback of the video.

5.9.3. Elapsed Time Until Video Playback

Table 5.16 presents the elapsed time until the video playback for the best test scenarios of each method.

Section – Test Method	**Best Test Scenario**	**Elapsed Time Until Video Playback [sec]**
Section 5.3 - Sliding Window Piece Selection Method	Sliding Window - 3 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	15 sec
Section 5.4 - Video Buffering Techniques	Sliding Window - 3 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	26 sec
Section 5.5 - Optimization of the Number of Downloading Peers and the Initial Choke/Unchoke Interval	Sliding Window – 3 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval - 6 sec	7.3 sec
Section 5.6 - Base Layer Assurance Mechanism	Sliding Window - 3 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval – 6 sec	2.4 sec
Section 5.8 - Alternative Piece Selection Method	Sliding Window - 5 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval - 5 sec Probability Schema – Any	3.0 sec < Time < 3.7 sec

Table 5.16: Elapsed time until the video playback for the best test scenarios of each method.

Comparing the obtained results in Table 5.16, it can be verified that the elapsed time until the video playback can be decreased considerably by increase of the number of downloading peers and adjusting the the initial choke/unchoke interval. However, the most reduction was obtained when introduced the base layer assurance mechanism in section 5.6, in which the value of the elapsed time until the video playback decrease to its minimum value of 2.4 seconds.

5.9.4. Delay Verified by the Re-Buffering Mechanisms

Table 5.17 presents the delay verified by the re-buffering mechanisms for the best test scenarios of each method.

Section – Test Method	Best Test Scenario	Delay Verified by the Re-Buffering Mechanisms [sec]
Section 5.3 - Sliding Window Piece Selection Method	Sliding Window - 3 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	-
Section 5.4 - Video Buffering Techniques	Sliding Window – 4, 5 Chunks 4 Downloading Peers Initial Choke/Unchoke Interval - 10 sec	0.6 sec
Section 5.5 - Optimization of the Number of Downloading Peers and the Initial Choke/Unchoke Interval	Sliding Window – 5 Chunks 6 Downloading Peers Initial Choke/Unchoke Interval – 1, 2 sec	0.6 sec
Section 5.6 - Base Layer Assurance Mechanism	Sliding Window - 5 Chunks 6 Downloading Peers Initial Choke/Unchoke Interval – 1 sec	0.2 sec
Section 5.8 - Alternative Piece Selection Method	Sliding Window - 5 Chunks 8 Downloading Peers Initial Choke/Unchoke Interval - 5 sec Probability Schema – Any	0.3 sec < Delay < 0.8 sec

Table 5.17: Delay verified by the re-buffering mechanisms for the best test scenarios of each method.

When compared the results in Table 5.17 appear to be very similar, but it can be verified that the best results were obtain with the introduction of the base layer assurance mechanism in section 5.6, in which the value of the delay verified by the re-buffering mechanisms decreased to 0.2 seconds.

5.9.5. Summary

The obtained results have shown that each method contributed differently to achieve the final result. While the video buffering techniques (introduced in section 5.4) and the alternative piece selection method (of section 5.8), had a very important contribution to the increment of the number of the layers received by the peers in the swarm; the adjustment of the number of downloading peers, the initial choke/unchoke interval and the base layer assurance mechanism (introduced in sections 5.5 and 5.6) were the main responsibles for the reduction of the delay verified between the availability

of the chunk files and playback of the video, the elapsed time until the video playback and delay verified by the re-buffering.

Chapter 6

Conclusions and Future Work

6.1. Conclusions

The BitTorrent overlay has shown to be capable of approaching the traditional IP Multicast based solutions for video distribution. When comparing the infrastructural requirements associated with both solutions, BitTorrent can easily be used over a shared distribution media like the Internet, while IP Multicast solutions are not supported on the Internet and require private network infrastructures for distributing content.

The main objective of this research was accomplished, comprising the implementation of a BitTorrent based H.264/SVC transmission system, complemented by the implementation and testing of several features using a simulation model. The results obtained are very promising, proving the usability of a Peer-to-Peer overlay to distribute live video content over the Internet.

The defined system not only solves scalability issues of the traditional video distribution systems when using the Internet as a distribution media, but also supports heterogeneity capabilities due to the usage of the scalable extension of H.264.

Enhancements like the an Incremental/Dynamic Torrent Files or the Sliding Window Piece Selection Method, provide unique capabilities to BitTorrent enabling it to handle real-time content distribution.

Other enhancements like the introduction of Video Buffering Techniques and Base Layer Assurance Mechanisms, added to the optimization of the Number of Downloading Peers and the Initial Choke/Unchoke Interval and combined with Alternative Piece Selection Methods, highlight the need for optimization of BitTorrent to handle live video content distribution.

In the tests performed using the simulation model, it was capable of delivering H.264 video content to a swarm of 100 peers, assuring that all the peers receive at least the H.264/SVC base layer. On

average, the delay between video availability in the server and its reception in a requesting peer reached 6.0 seconds while a value of 2.4 seconds was achieved for the time elapsed between a peer entering a BitTorrent swarm and starting to receive the requested content.

Finally, the impact of peer churn was evaluated through several tests using a simulation model for different peer churn rates. Those tests confirmed the independency of the quality of the video received by the peers and the BitTorrent swarm sustainability for different peer churn rates.

6.2. Future work

The currently defined P2P real-time video distribution system can still be further improved. This section lists some of the topics that deserve further study in order to optimize the system, in order to implement a video distribution system.

1. **Incremental/Dynamic Torrent File:** Development and implementation of a mechanism capable of updating the content of the Torrent file on real-time demand.
2. **Torrent Piece Size Optimization:** Investigate and optimize the choice of the Torrent Piece Size, according to the encoder definitions for generating chunk files in order to reduce the overhead on the padding of chunk files.
3. **Base Layer Assurance Mechanism Optimization:** Investigate and optimize the Base Layer Assurance Mechanism, in order to maximize the swarm sustainability and avoiding the sinking of the seeding peer.
4. **Alternative Piece Selection Method Optimization:** Investigate new Alternative Piece Selection Methods capable of maximizing the usage of upload and download bitrates on the peers and maximize the number of the most wanted file pieces in the swarm.
5. **Trial implementation:** Development and implementation of an experimental trial prototype of this system.
6. **Adaptation to other P2P overlays:** Investigate the possibility of usage of other well known P2P overlays to support real-time content distribution, based on the earnings of this study.

In conclusion Peer-to-Peer Real-Time Content Distribution is a research topic that is far from being ended and will be a topic for discussion and study in further researches, with many interesting aspects willing to be solved and researched, like which type of P2P overlay best suits the needs of a

real-time video distribution system or even if there is a way to enhance a P2P overlay to have control/management features for QoS.

Bibliography

[1] Cisco Systems, Inc, “Cisco Visual Networking Index: Forecast and Methodology, 2010–2015”, White Paper, Jun. 2011.

[2] J. F. Buford, H. Yu, E. K. Lua, “P2P Networking and Applications”, Morgan Kaufmann, 2009.

[3] PPLive, Available: www.pptv.com, Accessed: Sep. 20th, 2012.

[4] SopCast, Available: www.sopcast.org, Accessed: Sep. 20th, 2012.

[5] UUSee, Available: www.uusee.com, Accessed: Sep. 20th, 2012.

[6] P. Baccichet , T. Schierl, T. Wiegand, B. Girod, "Low-delay Peer-to-Peer Streaming using Scalable Video Coding," In Proc Packet Video 2007, Lausanne, Switzerland, pp. 173-181, 12-13 Nov.

[7] ITU-T Recommendation H.264 and ISO/IEC 14496-10, “Advanced Video Coding for Generic Audiovisual Services”, (MPEG-4 AVC), International Telecommunication Union–Telecommunication and International Organization for Standardization/International Electrotechnical Commission, Joint Technical Committee 1, Version 1: May 2003, Version 2: May 2004, Version 3: Mar. 2005, Version 4: Sept. 2005, Version 5 and Version 6: June 2006, Version 7: Apr. 2007, Version 8: Nov. 2007, Version 9: Jan. 2009, Version 10 and Version 11: Mar. 2009, Version 12 and Version 13: Mar. 2010, Version 14 and Version 15: June 2011, Version 16: Approved in Jan. 2012.

[8] R. Nunes, R. Cruz, M. Nunes, “Scalable Video Distribution in Peer-to-Peer Architecture,” 10ª Conferência sobre Redes de Computadores, CRC2010, 11 e 12 de Novembro de 2010, Universidade do Minho, Braga

[9] H. Schwarz, D. Marpe, T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard”, IEEE Trans. on Circuits and Systems for Video Technology, vol. 17, no. 9, pp. 1103-1120, Sep. 2007.

[10] D. Marpe, T. Wiegand, G. Sullivan, “The H.264/MPEG4 Advanced Video Coding Standard and its Applications”, IEEE Communications Magazine, Aug. 2006.

[11] J. M. Monteiro, "Quality Assurance Solutions for Multipoint Scalable Video Distribution over Wireless IP Networks - Dissertação para obtenção do Grau de Doutor em Engenharia Electrotécnica e de Computadores”, IST, pp. 9-16, Jun. 2010.

[12] J. M. Monteiro, C. T. Calafate, M. S. Nunes, "Evaluation of the H.264 Scalable Video Coding in Error Prone IP Networks", IEEE Transactions on Broadcasting, vol. 54, no. 3, pp. 652-659, Sept. 2008.

[13] J. M. Monteiro et al, “Multimedia Networking and Coding: From Capture to Display - Peer-to-Peer Video Streaming”, Submitted for reviewing to IGI Global for publication, Dec. 2010.

[14] B. Cohen, “The BitTorrent Protocol Specification”, Internet: http://www.bittorrent.org/beps/bep_0003.html, accessed: Nov. 13, 2011.

[15] J. Vieron, M. Wien, and H. Schwarz, JSVM-9 Software, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Doc. JVT-V203, Jan. 2007.

[16] Sourceforge, “Open SVC Decoder”, Available: "http://sourceforge.net/ projects/opensvcdecoder/”, [Accessed Jun. 5, 2010].

[17] P. Shah, J. F. Pâris, “Peer-to-Peer Multimedia Streaming Using BitTorrent”, University of Houston, 2007.

[18] S. Rahmanian, “IPTV Network Infrastrusture”, Huawei Technologies, Co. Ltd, Dec. 2008.

[19] Broadband Forum TR-126, “Triple-Play Services Quality of Experience (QoE) Requirements”, Dec. 2006. [Online] Available: ?http://www.broadband-forum.org/technical/download/TR-126.pdf?. [Accessed Aug. 31, 2012].

Appendixes

A.Implemented Simulation Framework

This section describes the simulation framework implemented in the tests performed in Chapter 5. The framework stands in a Matlab function (named P2PTV), which uses a discrete event simulation environment to simulate a BitTorrent swarm, including the messaging and signaling exchanged between peers. The block structure of the framework is shown in Figure A.1, which aggregates several versions of the P2PTV function.

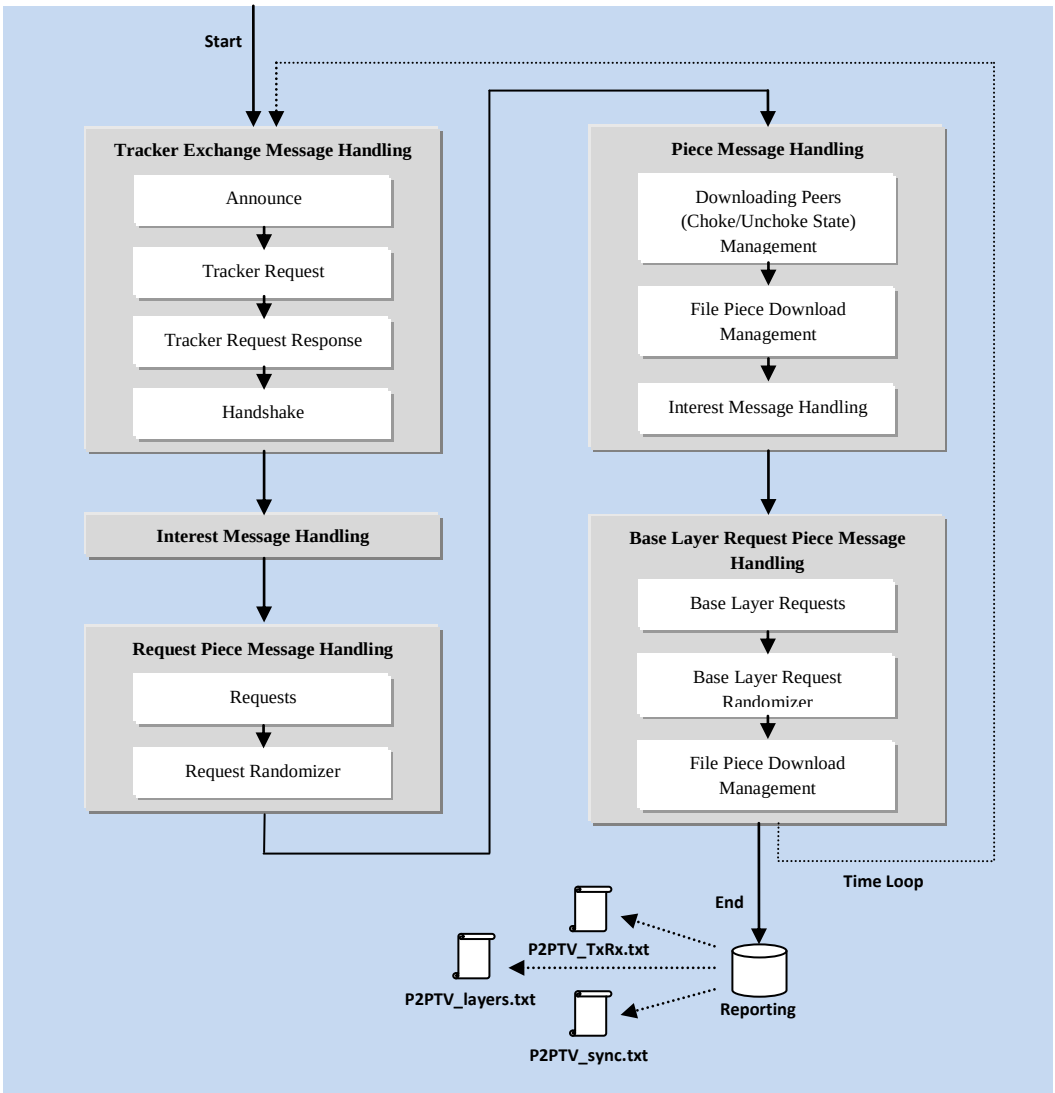

Figure A.1: Structure of the simulation framework considered in the tests performed in the work of this thesis.

The execution of the P2PTV function performs a complete simulation of the behavior of the BitTorrent swarm, starting with the entrance of the peers in the swarm, processing every message exchanged between peers and ending with the reception of the transmission layers by other peers.

The P2PTV function main structure consisted in a main block, a reporting block and five different blocks, each one responsible for a different group of tasks.

The main block was the responsible for managing and storing variable like time, swarm and peer status and controlling the main loop of the function by looping through other five blocks as time advances.

The five blocks consisted in the tracker exchange message handling block, which was responsible for handling every message and peer status interacting with the tracker; the interest message handling block, responsible for treating every piece interest message and the piece interest status of every peer; the request piece handling block, responsible for treating every requests for pieces messages on every peer; the piece message handling block, responsible for handling all the sending pieces messages and the pieces download; and the base layer request piece message handling block, for treating every base layer requests for pieces messages of every peer and the base layer pieces download.

Finally, the reporting block produced the reports used in the analisys performed along this thesis, with the values of the processed variables for each peer, transmission layer and messages exchanged. The produced reports consisted in three different text files with information of network bandwith usage (named P2PTV_TxRx.txt), transmission layers received by peer (named P2PTV_delays.txt) and delays and elapsed times for the each peer (named P2PTV_sync.txt).