A HTTP Streaming Video Server with Dynamic Advertisement Splicing

MD. SAFIQUL ISLAM

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:46

A HTTP Streaming Video Server with Dynamic Advertisement Splicing

Md. Safiqul Islam

March 21, 2010

Master of Science Thesis

Royal Institute of Technology (KTH)
School of Information and Communication Technology

Academic Supervisor: Professor Gerald Q. Maguire Jr., Royal Institute of Technology (KTH)

Industrial Supervisors: Ignacio Mas and Calin Curescu, Ericsson Research

Dedicated to my wife Monita for bearing with me - then, now and forever

Abstract

The Internet today is experiencing a large growth in the amount of traffic due to the number of users consuming streaming media. For both the operator and content providers, streaming of media generates most of its revenue through advertisements inserted in the content. One common approach is to pre-stitched (i.e. insert) advertisements into the content. Another approach is dynamic advertisement insertion, which inserts advertisements at run-time while the media is being streamed. Dynamic advertisement insertion gives operators the flexibility to insert advertisements based on context, such as the user's geographic location or the user's preferences. Developing a technique to successfully insert advertisements dynamically into the streaming media has several challenges, such as maintaining synchronization of the media, choosing the appropriate transport format for media delivery, and finding a splicing boundary that starts with a key frame. The details of these challenges are detailed in this thesis.

We carried out extensive research to find the best transport format for delivery of media and we studied prior work in an effort to find an appropriate streaming solution to perform dynamic advertisement insertion. Based upon this research and our study of prior work we identify the best transport format for delivery of media chunks, then propose, implement, and evaluate a technique for advertisement insertion.

Keywords: *HTTP Stream, MPEG-2 TS, MP4, Advertisements, Media Plane Management.*

Sammanfattning

Idag har internet mycket trafik på grund av att alltfler servrar erbjuder högkvalitativa videon som strömmas till internetanvändare. Både för operatörer och leverantörer av sådan innehåll genererar direktuppspelning mest intäkter genom annonser som lagts till i videon. Det är väldigt vanligt att lägga till annonser i videon genom att sy in dem i videofiler. En annan metod är att lägga till annonser dynamiskt. Det betyder att resulterande videofilen genereras medan den blir strömmad till användare. Att sätta in annonser dynamiskt har som fördel för operatörer att välja reklam beroende på kontexten, såsom användarens position eller preferenser.

Det är utmanande att utveckla den teknik som krävs för att kunna sätta in annonser dynamiskt i strömmade videofiler. Till exempel är det viktigt att tänka på följande: synkronisering av strömmad innehåll, val av lämplig transportformat för videoleveransen och gränsen för skarvning (så kallad splicing boundary). Detaljerna kring denna teknik finns i denna avhandling.

Vi har forskat på att hitta det bästa transportformatet för videoleverans och vi har studerat relevant arbete som gjorts tidigare för att hitta en lämplig mekanism för dynamisk annonsinsättning. Baserat på vår forskning och studerande av tidigare arbeten har vi klassificerat det bästa formatet för leveransen av videostycken, implementerat och evaluerat en teknik för annonsinlägg.

Acknowledgements

I have no words to express my gratitude to my academic supervisor and examiner **Professor Gerald Q. Maguire Jr.** of Royal Institute of Technology. Throughout my work, he has been helping me with critical reviews, providing detailed background knowledge. His personal involvement in the project made it possible to write a successful thesis and I am immensly debted to him for this.

I would like to show my heartiest gratitude to my supervisor **Ignacio Mas** of Ericsson Research for his continuous encouragement, inspiration and technical advices that helped me to look beyond the boundaries. I am also thankful my another superviser **Calin Curescu** of Ericsson Research for asking me lots of 3 questions (what, when, and why) that influenced me to look deeper into that area. I would like to thank my colleague **Peter Woerndle** for his extensive support for helping me during the analysis process.

I want to express my respect and gratitude to my parents, especially my mother for allowing me to do my masters studies in KTH and maintaing faith in me. Thanks to all my friends who have inspired me, especially **Rezaul Hoque**, **Raisul Hassan**, **Riyadh-ul Islam**, and **Ferdous Alam** for believing in me and keeping me alive in this frozen land.

Finally, I would like to thank my wife **Zannatul Naim Monita** for providing my best inspiration and support, because without you my love, it would not be possible at all.

Stockholm, March 21, 2010

Safiqul Islam

Contents

List of Figures

List of Tables

Listings

List of Acronyms and Abbreviations

CDN	Content Delivery Network
CIM	Context Information Module
DI	Discontinuity Indicator
DTS	Decoding Time Stamp
FMP4	Fragmented MP4
HD	High Definition
HTTP	Hyper Text Transfer Protocol
MDL	Media Delivery Logic
MP4	MPEG 4 Part 14
MPEG	Moving Picture Expert Group
MPM	Media Plane Management
PAIL	Personalized Advertisement Insertion Logic
PAT	Program Association Table
PMT	Program Map Table
PES	Packetized Elementary Stream
PCR	Program Clock Reference
PTS	Presentation Time Stamp
SIP	Session Initiation Protocol
STB	Set Top Boxes
TS	Transport Stream
URL	Uniform Resource Locator

Chapter 1

Introduction

1.1 Motivation

In recent times, streaming has been a widely used approach for media delivery. Delivering media over Hypertext Transfer Protocol (HTTP) [1] has been popular for content providers since the introduction of HTTP. Additionally, classic streaming protocols (such as RTP) [2] are popular for audio and video streaming via the Internet. In recent years, many content providers have migrated from classic streaming protocols to HTTP. This has been driven by four factors [3]: (1) HTTP download is less expensive than media streaming services offered by Content Delivery Networks (CDN) and hosting providers, (2) HTTP protocol can generally bypass firewalls; as most firewalls allow return HTTP traffic from TCP source port 80 through the firewall - while most firewalls block UDP traffic except for some specific ports, (3) HTTP delivery works with any web cache, without requiring special proxies or caches, and (4) it is easier and cheaper to move HTTP data to the edge of the network, i.e., close to users, than for other protocols. As a result, HTTP based adaptive streaming is the current paradigm for streaming media content. In addition to the advantages noted above for HTTP, this approach has gained immense popularity due to a shift from delivering a single large file to the delivery of many small chunks of content.

It is well known that advertising is most effective when the advertisements are relevant to the viewer. If operators are able to deliver advertisements dynamically during online streaming by inserting advertisements based on context, such as geographic location or the user's preferences, then the operator can provide relevant advertisements to the target audience. However, finding the appropriate splicing boundaries is a challenging task when inserting advertisements into streaming video. If done successfully, advertisement insertion in HTTP based streaming media can generate revenue for both the operator and content

providers[4].

Ericsson's Media Plane Management (MPM) reference architecture [5] works as a mediator between operators and Internet content providers to optimize and manage media delivery. Additionally, in this framework the operator acts as a mediator between the content providers and end users by offering intelligent media delivery. The architecture describes the requirements for advertisement insertion techniques to be implemented in a media server. One of the key requirements of the MPM project is to select the best transport format for media delivery; to enable an advertisement's contents to be fetched, synchronized, and spliced into the stream being delivered to the target user.

1.2 Goal

The main goal for this thesis project was to implement a streaming server that will fetch streaming contents and advertisements from their respective content servers and to cleverly stitch the advertisements into the media stream *before* feeding the stream to a client. The project began with a study of existing streaming solutions using HTTP with an adaptive streaming extension. A secondary goal was to find the most appropriate transport stream format. The overall goal is to propose, implement, and evaluate a solution based upon the best transport format for delivery of chunks of streaming media based upon an advertisement insertion technique that was proposed as part of this thesis project.

1.3 Research Questions

Based on the main thesis goal (and the secondary goal) mentioned in the previous section, here are some research questions that this thesis project attempts to answer. This thesis project focused on the following four questions:

Question 1 To deliver the media chunks an appropriate container is required. Which is the most appropriate container?

Question 2 Can the solution be sufficiently portable that it will support different vendors' end-devices, such as Motorola's set top boxes (STBs), SONY's Play Station 3, and Apple's iPhone?

Question 3 How can we maintain the continuous flow of a stream including the advertisement? This means that it is very important to find out the proper splicing boundaries for advertisement insertion in order to maintain the stream's continuity.

Question 4 Can the solution be implemented successfully on a constrained server, while delivering media to the client within the appropriate delay bound?

1.4 Thesis Outline

This thesis is organized so that the reader is first presented with the appropriate theoretical background before delving into the details. Chapter 2 gives an overview of streaming, CODECs, containers, and MPM architecture. Chapter 2 also delimits the scope of our thesis in terms of the Ericsson MPM architecture. Several existing streaming approaches are described in chapter 3 in order to understand their transport techniques. Our proposed solution is given in chapter 4. Chapter 5 presents an evaluation of the proposed solution. Finally, chapter 6 summarizes our conclusions, research findings, and offers some suggestions for further improvements.

Chapter 2

Background

2.1 Streaming

Media streaming is a process to deliver continuous media such as video or audio to a receiver as a continuous stream of packets. Due to this stream of packets, the receiver does not need to download the entire file before starting to play (or render) the media. Media delivery is currently based on three general methods: traditional streaming, progressive download, and HTTP chunk based streaming. The following subsections will describe these three streaming techniques.

2.1.1 Traditional Streaming

Using a traditional streaming protocol media is delivered to the client as a series of packets. Clients can issue commands to the media server to play (i.e., to send a media stream, to temporarily suspend this stream (i.e., to pause the media), or to terminate the media stream (i.e., to teardown the media stream). One of the standard protocols for issuing these commands is the Real Time Streaming Protocol (RTSP)[6].

One of the traditional streaming protocols is the Real-Time Transport Protocol (RTP)[2]. Traditional streaming is based on a stateful protocol (RTSP) where the server keeps track of the client's state. However, Microsoft used the stateless HTTP protocol for the streaming - this is officially known as MS-WSMP[7]. To keep track of the state of the client they used a modified version of HTTP.

2.1.2 Progressive Download

One of the most widely used methods of media delivery on the web today is progressive download. This is basically a download of a file from the web server,

but with the client starting to play the media contents of this file before the file is completely downloaded. Unless the media stream is terminated, eventually the entire file will be downloaded. In progressive download, downloading continues even if the user pauses the player. The Internet's most popular video sharing website - YouTube – uses progressive download [8].

2.1.3 Adaptive streaming - HTTP based delivery of chunks

Adaptive streaming is based upon progressive download of small fragments of the media, but the particular fragments that are downloaded are chosen based upon an estimate of the current network conditions. Each of these fragments is called a chunk. Thus "adaptive streaming" is not actually streaming the media content, but instead it is an adaptive version of HTTP progressive download!

However, adaptive streaming is actually *adaptive* since once the input media was split into a series of small chunks, each of these chunks can be encoded into one or more of the desired delivery formats for later delivery by an HTTP server. Each chunks can be encoded at several bit rates (i.e., using different CODECs (see next section) and different parameters), hence the resulting encoded chunks can be of different sizes. The client requests the chunks from the server and downloads the chunks using the HTTP progressive download technique. The actual adaptation is based upon the client choosing a particular version of each chunk. The version of the chunk that is requested is based upon the client's estimate of the current network conditions and the load on the server (i.e., if the server is heavily loaded or the network seems congested, then the client requests a smaller instance of the chunk, otherwise it can request a larger version of the chunk - potentially providing higher resolution or greater color fidelity). After a chunk is downloaded to the client, the client schedules the play out of the chunk in the correct order – enabling the user to watch a seamless video (and or hear a seamless audio track). The client can also play the available chunks in any order that the user would like, allowing "instant" replays, freeze frame, and other video effects.

This thesis project focuses on adaptive streaming because it provides the following benefits to the user:

- It provides fast start-up and seek times with in a given item of content by initiating the video at the lowest video rate and later switching to a higher bit rate.
- There is no disconnection, buffering, or playback stutter problem.
- It provides seamless bit rate switching based on network conditions.
- It also provides the user with seamless video playback.

2.2 CODECs

A coder/decoder (CODEC) is used to encode (decode) video or audio. Many CODECs are designed to compress the media input produced by the source in order to reduce the data rate needed for the media or to reduce the storage spaced required to store the resulting media or some combination of these two. A CODEC can be lossless, i.e., the decoded data is identical to the original data, or the CODEC can be lossy, i.e., the decoded data is not identical to the original data. Lossy coding schemes can achieve higher compression ratios (the ratio of the output size to the input size is much less than 1); however, there will be some loss in quality. In recent years lossy perceptual-based CODECs have become popular as they minimize the number of bits used to encode the original content yby eliminationg content that is least relevant to the perception of the viewer.

As we are concerned with video, we will describe in the next paragraphs the two most popular standard video CODECs. There are also many proprietary video CODECs, but we will not consider them in the scope of this thesis project.

2.2.1 MPEG-2

The Moving Picture Expert Group (MPEG) 2 standard [9][10] is a popular CODECs for compressed video. A MPEG-2 video sequence can be divided into groups of pictures. Within the group of pictures, each picture is referred to as frame. Pictures can also be divided into a slice. A group of four blocks is known as macroblock. A block is the smallest group of pixels that can be displayed on the screen. Figure 2.1 illustrates the relationships between these entities.

The MPEG standard defines three types of pictures:

Intra Pictures (I-Pictures) I-pictures are encoded using only the information that is present in the picture.

Predicted Pictures (P-Pictures) P-pictures are encoded while exploiting the information from the nearest previous I-pictures or P pictures. The technique is normally known as forward prediction. P-pictures provide higher compression than I-pictures.

Bidirectional Pictures (B-Pictures) B-pictures use both the previous and subsequent pictures for reference. This is known as bi-directional prediction. These pictures provide the highest compression, because the compression can take advantage of both the past and future contents. However, the computational cost of encoding is higher than for I-pictures or P pictures.

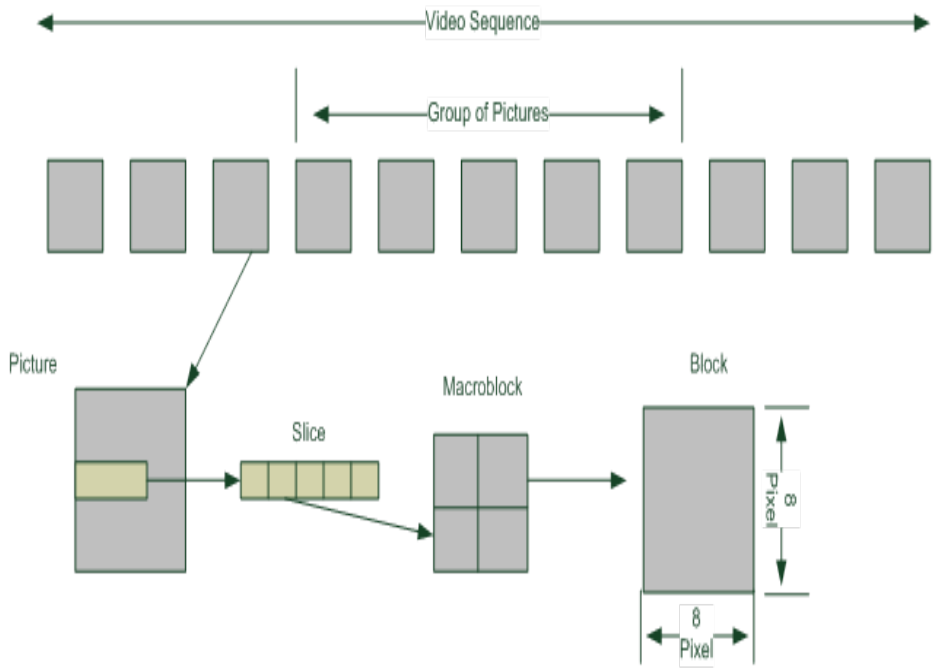


Figure 2.1: MPEG-2 Video Structure, adapted from [9]

2.2.2 MPEG-4 Part 10

H.264, also known as MPEG-4 part 10, is a standard jointly developed by the ITU-T Video Coding experts and the ISO/IEC Picture Expert Group [11][12]. The standard deals with error resilience by using slicing and data partitioning. The main advantage of this standard as compared to MPEG-2 is that it can deliver better image quality at the same bit rate for the compressed stream or a lower bit rate while offering the same quality [13]. Compared to earlier standards, H.264 includes two additional slice types: SI and SP [14]. In general, SI and SP slices are used for synchronization and switching. These slice types are used while switching between similar video contents at different bit rates and for data recovery in the event of losses or errors. Arbitrary slice ordering offers reduced processing latency in IP networks [15] as packets may arrive out of order.

H.264 has gained popularity in several application areas, including [13][16]:

- High Definition (HD) DVDs,
- HD TV broadcasting,
- Apple's multimedia products,
- Mobile TV broadcasting, and
- Videoconferencing.

The H.264 video encoding process takes video input from a source and feeds it to prediction, transform, and encoding processes to produce a compressed

bitstream. The encoder processes a video frame in units of macroblocks and forms a prediction of the macroblock with the information from either the current frame (for intra prediction) or from other coded or transmitted information (known as inter prediction). The prediction method of H.264 is much more flexible and accurate than the prior standards. The transform process produces a quantized transform of coefficients as its output [17]. Finally, the encoding process produces the compressed bitstream.

The decoding process is the reverse of the encoding process. It feeds the compressed stream to a decoder, does an inverse transform, and reconstructs the pictures to generate video output. The entire process is illustrated in figure 2.2.

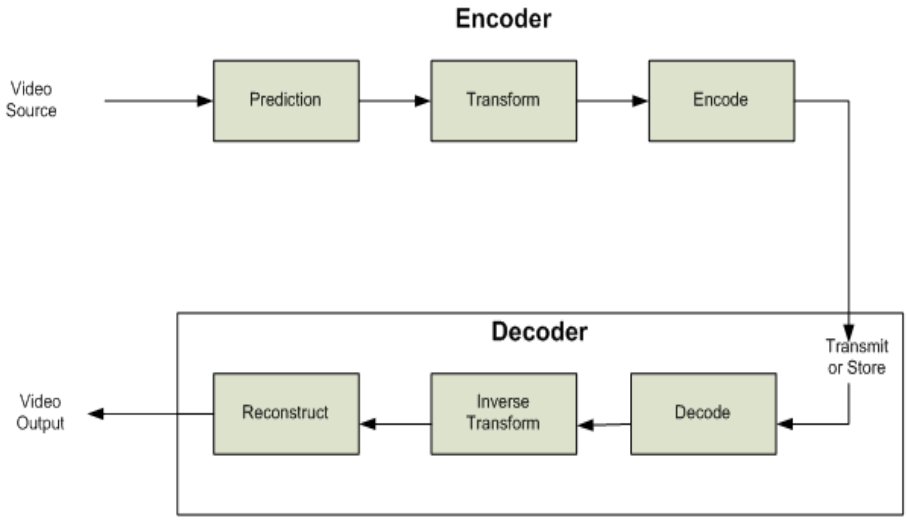


Figure 2.2: H.264 video encoding and decoding process, adapted from [13]

2.3 Container Formats

A container format is a wrapper that contains information such as: video, audio, or subtitles. A container format is also known as a meta format as it stores both the data itself along with additional information. The following two sub-sections describe two popular container formats used for streaming audio and video.

2.3.1 MPEG-2 Transport Stream

An MPEG-2 transport stream (MPEG-2 TS) multiplexes various Packetized Elementary Streams (PESs) into a single stream along with synchronization information. A program is formed from the PES packets from the elementary streams. MPEG 2 defines a transport stream for storing or transmitting a program. Logically a transport stream is simply a set of time-multiplexed packets

from several different streams [10][18][19][20]. The overall transport stream format is shown in Figure 2.3. The packet format and stream generation process are described in the following paragraphs.

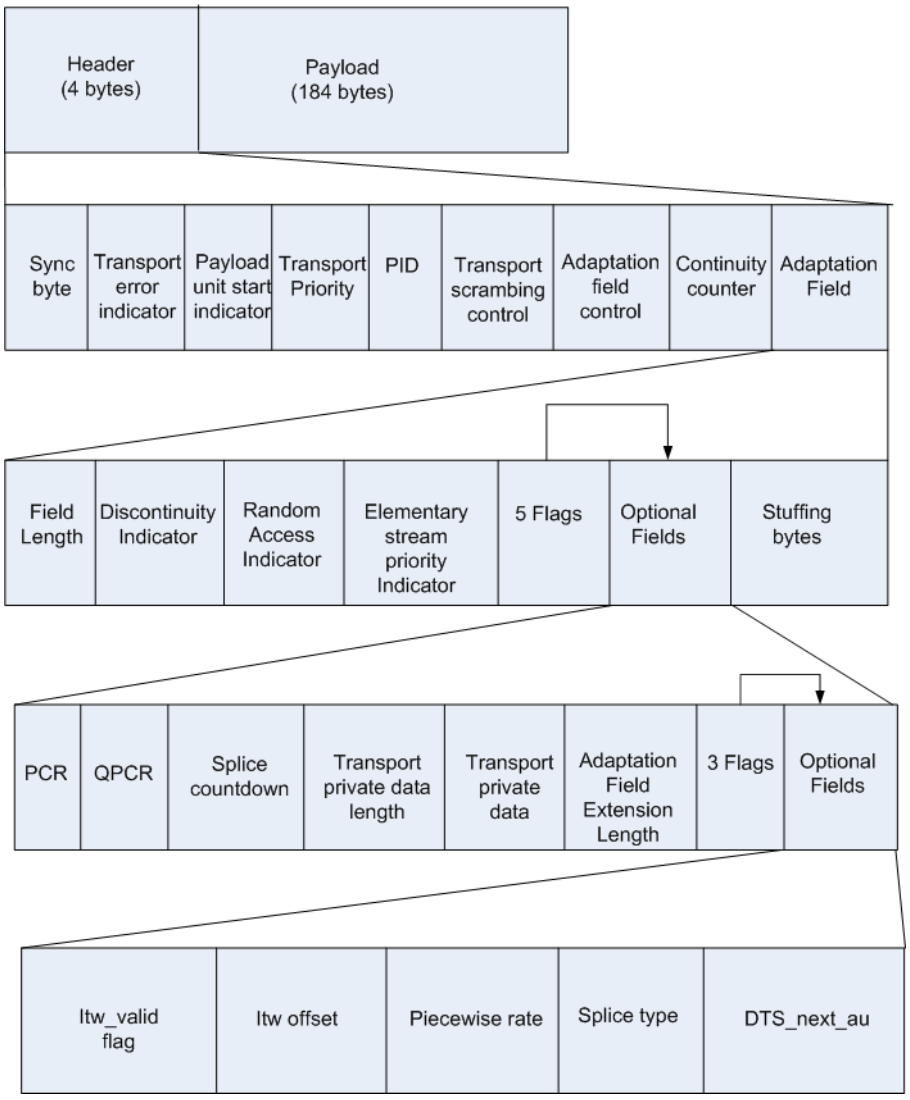


Figure 2.3: Overall Transport Stream, adapted from [18]

2.3.1.1 Packetized Elementary System

An elementary stream is a compressed form of an input source, such as video or audio. A packetized elementary stream (PES) is formed by packetizing the elementary streams into fixed size or variable size packets. Each PES packet consists of a PES header and payload. Figure 2.4 illustrates the packet format of a PES.

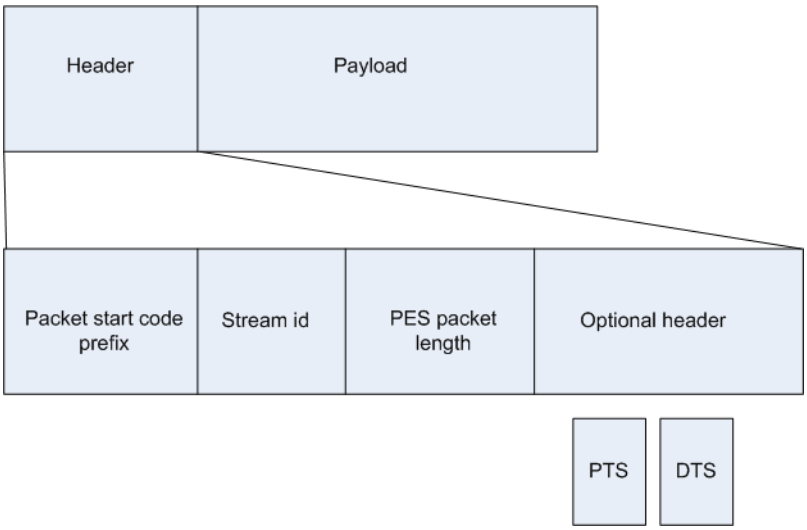


Figure 2.4: PES Packet Header, adapted from [19]

The PES header begins with a start code prefix (three bytes containing the value 0x000001). The Stream ID is followed by the PES packet length and an optional header. This stream ID (1 byte) specifies the type of stream. The PES packet length (2 bytes) defines the length of the packet.

2.3.1.2 MPEG-2 TS packet Format

MPEG-2 TS uses a short, fixed length packet of 188 bytes; consisting of 4 bytes of header, an optional adaptation field, and payload as shown in Figure 2.5.

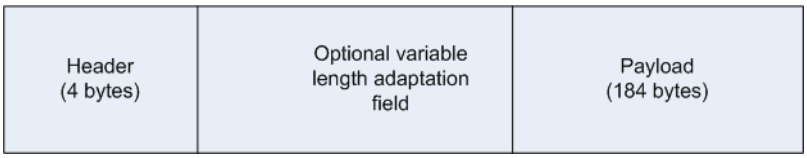


Figure 2.5: MPEG-2 TS packet, adapted from [19]

Figure 2.6 shows the MPEG-2 TS header. The fields in this header are:

- A sync byte used for random access to the stream.
- The transport error indicator provides error indication during transport.
- The payload unit start indicator is followed by the transport priority, this indicates the presence of a new packet.
- A Program ID (PID) allows identification of all packets belonging to the same data stream. Different streams may belong to different programs or to the same program. PIDs are used to distinguish between different streams.

- The scrambling mode used for the packet's payload is indicated by the transport scrambling control field.
- The continuity counter field (CC) is incremented by one for each packet belonging to the same PID.
- The presence of the adaptation field in the packet is indicated by the adaptation control field.

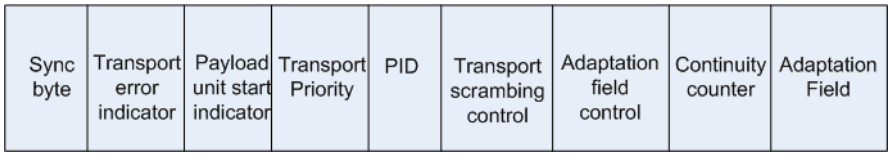


Figure 2.6: MPEG-2 TS header, adapted from [19]

Figure 2.7 shows the contents of the adaptation field. The sub-fields of the adaption field are:

- Field Length indicates the number of bytes following.
- Discontinuity indicator indicates whether there is a discontinuity in the program's clock reference.
- Random access Indicator indicates whether the next packet is a video frame or an audio frame.
- Elementary stream indicator is used to distinguish the priority of different elementary streams.
- Stuffing bytes in the adaptation field are used to pad the transport packet to 188 bytes.

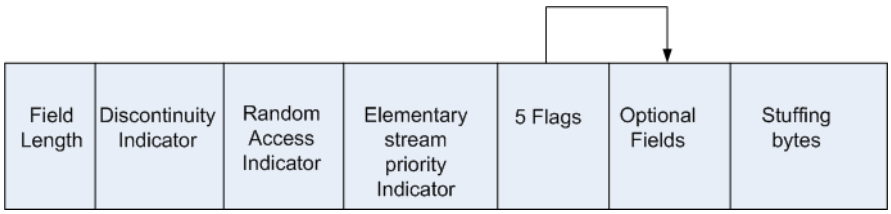


Figure 2.7: Adaptation Field, adapted from [19]

Figure 2.8 illustrates the format of the optional field. The fields are:

- PCR flag indicates the presence of a program clock reference (PCR).

- The OPCR flag represents the presence of an original program clock reference (OPCR).
- Splice countdown (8 bits) is used to identify the remaining number of TS packets of the same PID until a splicing point is reached.
- The number of private data bytes is specified by the Transport private data. The number of bytes of the extended adaptation length is indicated by Adaptation field extension length.

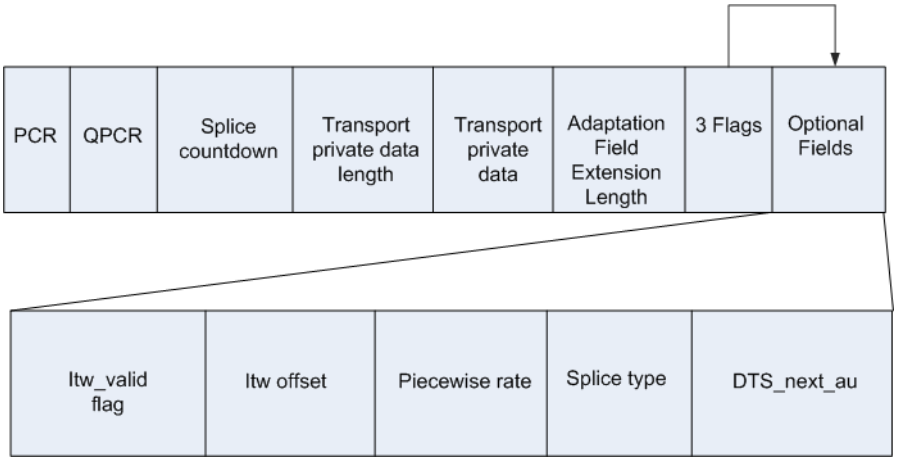


Figure 2.8: Optional field, adapted from [19]

2.3.1.3 Transport Stream Generation

A PES is the result of a packetization process and the payload is created from the original elementary stream. The transport stream is created from the PES packet as shown in Figure 2.9.

2.3.1.4 Synchronization

Synchronization in achieved through the use of time stamps and clock references. A time stamp indicates a time according to a system time clock that a particular presentation unit should be decoded and presented to the output device. There are two kinds of time stamps: Presentation Time Stamp (PTS) and Decoding Time Stamp (DTS). PTS indicates when an access unit should be displayed in the receiving end. In contrast, DTS indicates when it should be decoded. These time stamps (if present) are placed into the PES packet header's optional field. A clock reference is included in a transport stream through a Program Clock Reference (PCR). The PCR provides synchronization between a transmitter and receiver; it is used to assist the decoder to present the program on time.

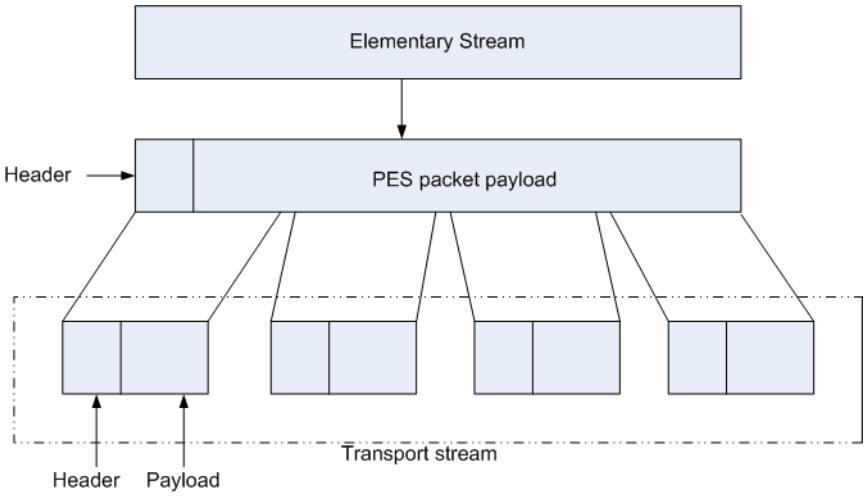


Figure 2.9: Transport Stream Generation, adapted from [19]

2.3.1.5 Program Specific Information

Program specific information (PSI) transport packets enable the decoder to learn about the transport stream. The PSI is a specialized TS stream that contains program descriptions and the assignments of PIDs and packetized elementary streams to a program. The PSI transport stream consists of the following:

- Program Association Table (PAT),
- Program Map Table (PMT),
- Network Information Table, and
- Conditional Access Table.

The PMT contains the PID for each of the channels associated with a particular program. The PAT is transmitted in transport packets with PID 0 - this table contains a list of all programs in the transport stream along with the PID for the PMT for each program. The details of header information of PAT and PMT can be found in appendix A. Figure 2.10 illustrates the relation between PAT and PMT and more details can be found in [21]. In this thesis we can ignore both the Network Information Table and Conditional Access Table because they are not relevant to us.

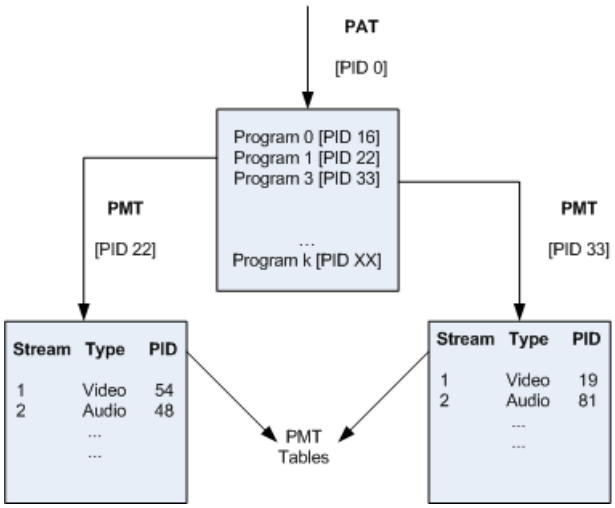


Figure 2.10: Relation between PAT and PMT table, adapted from [21]

2.3.2 MPEG-4 Part 14

MPEG-4 part 14 is an ISO standard multimedia container format specified as part of MPEG 4[22]. In general, this format is used to store audio and video streams as well as subtitles and still images. This format is frequently used for streaming over the internet and is referred to as "MP4" (the file extension of this format is ".mp4").

A MP4 file consists of a moov box that contains time-sample-metadata information. The moov box can be placed either at the beginning or at the end of the media file. The Media Data Container box (mdat) contains the audio and video data. Figure 2.11 shows the MP4 file format (see [23]).

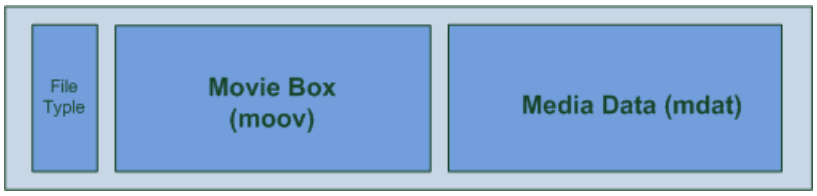


Figure 2.11: MP4 file format, adapted from [23]

In smooth streaming [8], Microsoft uses fragmented MP4 (FMP4) [24] for streaming. Section 3.2 describes these concepts in detail.

2.4 Content Delivery Networks

A content delivery network (CDN) consists of a group of computers that are situated between content providers and content consumers. In a CDN, contents

are replicated to a distributed set of content servers so that consumers can access a copy of the content from the "nearest" content server*. A CDN offers a number of advantages over a traditional centralized content server because content replication alleviates the bottleneck of a single server[25], allows increased scalability, and increases the robustness and reliability of the system by avoiding a single point of failure (*once* the content has been distributed to the CDN).

2.4.1 Amazon Cloud Front

Amazon's CloudFront [26] is a web service for content delivery using a global network of web servers. Amazon's CloudFront caches copies of content close to the end user. A request generated from a client is automatically routed to the nearest web server. All the objects are managed by an Amazon S3 bucket [27], which stores the original files, while CloudFront is responsible for replicating and distributing the files to the various servers. By distributing the content to server close to the requestor, Amazon CloudFront reduces the latency of downloading an object. In addition, the end user pays only for the data transfer and requests that they initiated.

2.4.2 Akamai HD Network

The Akamai HD Network is an on-line high definition (HD) video delivery solution [28]. It supports delivery of live and on demand HD quality video. Together with Akamai's HD Edge Platform solution, content is replicated close to the consumer. HD Adaptive Bitrate Streaming provides fast video start up (i.e., the time between the user selecting content and this content being displayed is short) and uninterrupted playback at HD quality. The Akamai HD network also supports an HD Authentication feature to ensure authorization for each Flash player before delivering content.

2.5 Advertisement Insertion and Detection

Advertisement based revenue has always been a major components of business models for distributing contents. This section describes advertisement insertion and detection techniques for placing advertisements in the bit stream. However, advertisement detection techniques are outside the scope of thesis.

2.5.1 Advertisement Insertion

Advertisement insertion techniques described in [29] have considered the following parties for the process: Content providers, Network operators, and Clients.

*Here "nearest" refers to nearness from a network topology and delay point of view. The nearest server is the server that can deliver the desired content in the shortest amount of time.

Content providers locate one or more advertisement insertion points and send the encrypted media to the network operators. Subsequently, network operators decrypt the encrypted media, and then using an advertisement inserter module, the network operators insert advertisements. Finally, network operators encrypt the media with the included advertisements and send it to one or more clients.

Advertisers rely upon the advertisement insertion points selected by the content provider. This thesis work does not focus on the advertisement insertion points selected by the content providers or the encryption and decryption of the media; rather, this thesis work focuses on the advertisement inserting module.

The Society of Cable and Telecommunications Engineers (SCTE), introduced the SCTE35 [30] standard. This standard was published by the American National Standards Institute (ANSI). The standard describes the timing information and upcoming splicing points. The standard defines two types of splice points;

- **In point splicing** defines the entry of a bitsream.
- **Out point splicing** defines the exit from the bitstream.

A splice information table is used for defining the splice events and this table is carried by PID values that reside in a PMT table.

Schulman [31] proposed a method for digital advertisement insertion in video programming. The method describes using externally supplied programming that contains embedded cue tones (pre-roll cue and roll cue), are detected prior to converting the analog video to digital video. In [32], Safadi proposed a method for digital advertisement insertion in a bitstream by providing a digial cue message – corresponding analog cue tones. An advertiser inserts the advertisement after detecting the digital cue message. A method for non-seamless splicing of transport streams is described in [33].

2.5.2 Advertisement Detection

This thesis project did not focus on advertisement detection technique, there are several exisiting methods for detecting advertisement. Peter T. Barrett describes local advertisement detection in [34]. His patent describes an insertion detection service, where a splice point in the video is detected in order to identify where the advertisement has been inserted through the following conditions (details can be found in [34]):

- Forced quantization match
- Video frame pattern change
- Timing clock change
- Picture group signaling change
- Insertion equipment signature
- Bit rate change

- Extended data service discontinuity
- Audio bit rate change

Jen-Hao et al. has described telvision commercial detection in news program videos in [35]. More information regarding advertisement detection, along with advertisement signature tracking is described in [36].

2.6 Ericsson's Media Plane Management Reference Architecture

Ericsson's media plane management project proposed a media plane management reference architecture which works as a mediator between operators and Internet content providers to optimize managed media delivery [5]. (See figure 2.12.) Their goals were to:

- Combine a smart storage and caching solution in both the Internet and network operator's network.
- Adapt the Internet content consumption based on the client device's capabilities.
- Allow personalized advertisement insertion.
- Provide access control and digital right management facilities.

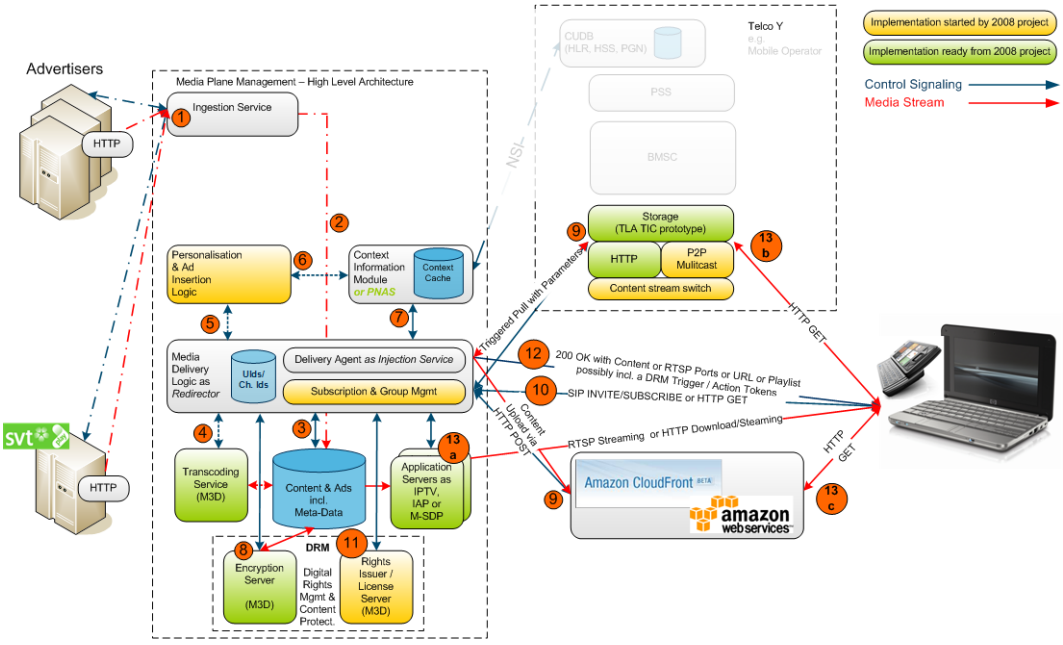


Figure 2.12: Ericsson's MPM Architecture, taken from [5] (Appears with permission of the MPM project.)

2.6.1 Overview

The MPM architecture allows content providers and advertisers to upload their files along with the relevant metadata information. After the content is uploaded, then the Media Delivery Logic (MDL) sends this content to a transcoding service that transcodes the content to different formats to be later used depending upon the user's context. After this, Personalization and Ad Insertion Logic (PAIL) together with MDL selects the best matching advertisement(s) for a given user. The Context Information Module (CIM) gathers context information from various sources such as the Home Subscriber Server. Based on the popularity of the contents and context information (such as the user's location), contents are uploaded to the operator's network used by the target users and to Amazon's CloudFront[26].

The main key idea of the MPM architecture is to use several storage and caching locations to minimize costs while maximizing the quality of the delivered media content. The following storage components were considered:

- Internal Storage - Contents and Advertisement database (DB),
- Amazon's CloudFront, and
- Operator provided storage.

The client initiates their media consumption by making a request either via Session Initiation Protocol (SIP) or HTTP. The SIP interface will be used if the request comes from an IP Multimedia Subsytem (IMS) domain.

2.7 Thesis Overview

This thesis project was conducted as part of the MPM project. The thesis project focuses specifically on the segmentation of the video contents and advertisement insertion at splicing points, i.e., an advertisement can be spliced in between two segments of the original contents.

Together the MDL and PAIL select the best matching advertisements based on information provided by the CIM. Two approaches have been described: splicing by the client or splicing by a server [5]. The first approach delivers the playlist directly to the client. This approach allows the client to flexibly fetch the media items itself, thus optimal transport paths can be used. However, the major disadvantage of this approach is that since the client fetches the information by itself an improperly secured client could allow users to skip the advertisements and play only the contents.

The second approach uses a streaming server to fetch all the media files from their respective locations, splice them together while inserting the advertisement, and then serving the result as a single media stream to the client. The disadvantage of this approach is it has poor scalability, because the streaming

server performing the splicing for each user resides in the MPM framework and the hardware for this splicing has to be provided by someone other than the end user.

Because Ericsson's main focus is on the communications infrastructure, rather than the handset, this thesis project has adopted the second approach and focuses on segmenting the content at splicing boundaries and inserting an advertisement prepared by MDL and PAIL. The core design of our system is the proxy streaming server with advertisement insertion logic. When the client requests a video, the streaming server communicates with the node that maintains the video chunks along with advertisement and splices the video. After this the media will be delivered to the client as a single HTTP resource. Figure 2.13 shows an overview of the splicing and advertisement insertion logic.

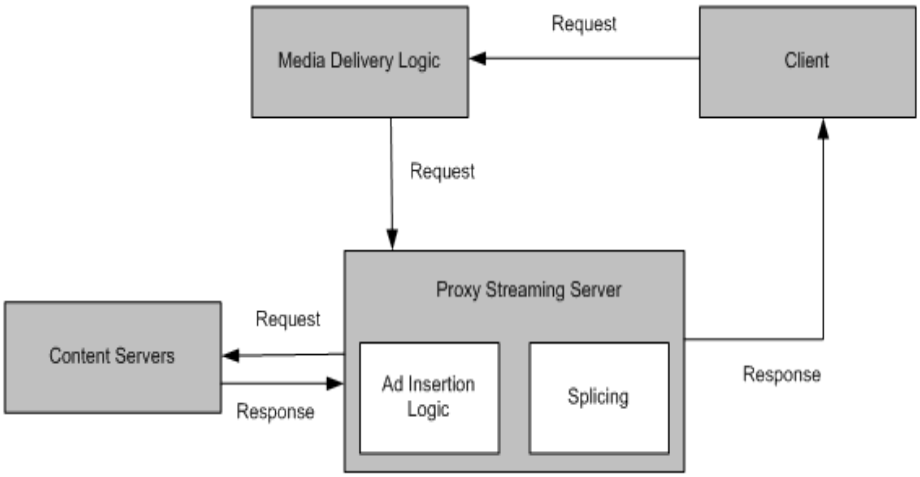


Figure 2.13: Overview showing the context of the splicing and advertisement insertion logic

Chapter 3

Related Work

This chapter discusses two existing approaches, specifically the Apple and Microsoft streaming approaches. These solutions are relevant to our thesis because they stream content to the client *after* segmentation. These two solutions are also widely used; as they are bundled with the operating systems from these two vendors. We will focus on identifying the key components of these two approaches. In addition, we will describe some existing advertisement insertion technologies.

3.1 Apple Live Streaming

In [37], Apple describes their HTTP live streaming solution. Their solution takes advantage of MPEG-2 TS and uses HTTP for streaming. Their solution consists of three components: Server Component, Distribution Component, and Client Software.

- The Server Component handles the media streams and digitally encodes them, then encapsulates the result in a deliverable format. This component consists of an encoder and a stream segmenter to break the media into a series of short media files.
- The Distribution Component is simply a set of web servers that accept client requests and deliver prepared short media files to these clients.
- Client Software initiates a request and downloads the requested content and reassembles the stream in order to play the media as a continuous stream at the client.

Figure 3.1 shows the resulting simple HTTP streaming configuration. The encoder in the server component takes and audio/video stream and encodes and

encapsulates it as MPEG-2 TS, then delivers the resulting MPEG-2 TS to the stream segmenter.

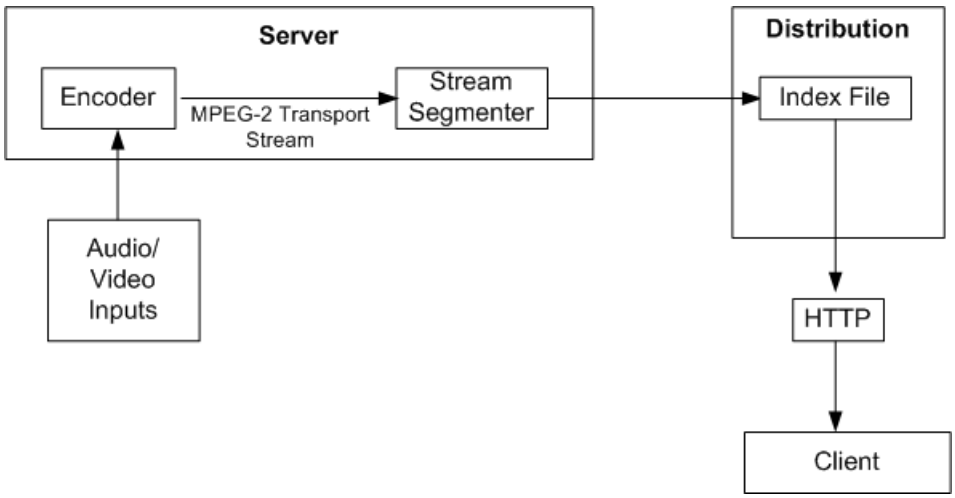


Figure 3.1: HTTP streaming configuration, adapted from [37]

The stream segmenter reads the transport stream and divides the media into a series of small media files. For broadcast content, Apple suggests placing 10 seconds of media in each file [37]. In addition to the segmentation, the segmenter creates an index file containing references to the individual media files. This index file is updated if a new media file is segmented. The client fetches this index file, then requests the URLs specified in the index file, finally the client reassembles the stream and plays it.

Apple provides three modes for configuring encryption in the media stream segmenter to protect the contents [37]. In the first mode, the segmenter inserts the URL and an encryption key in the index file. This single key is used to encrypt all the files. In the second mode, the segmenter generates a random key file and saves it to a location, then add this reference to the index file. A key rotation concept is used in the third mode, where the segmenter generates a random key file of n keys, stores this file and references it in the index file, then the segmenter cycles through this set of keys as it encrypts each specific file. The result is that each file in a group of n files is encrypted with a different key, but the same n keys are used for the next n files.

While using a unique key for each file is desirable, having to fetch a key file per segment increase the overhead and load on the infrastructure. Apple has submitted their approach for HTTP live streaming to the IETF[38].

3.2 Microsoft Smooth Streaming

Microsoft introduced "Smooth Streaming" [8], based on an adaptive streaming extension to HTTP. This extension was added as a feature of their Internet Information Services (IIS) 7 - web server. Smooth Streaming provides seamless bit rate switching of video by dynamically detecting the network conditions. They have used an MP4 container for delivering the media stream. The MP4 container is used as both a disk (file) format for storage purposes and a wire format for transporting the media.

Each chunk is known as an MPEG-4 movie fragment. Each fragment is stored within a contiguous MP4 file. File chunks are created virtually upon a client's request. However, the actual video is stored on disk as a full length MP4 file. A separate MP4 file is created for each bit rate that is to be made available.

The complete set of protocol specifications are available at [39]. Microsoft's Silverlight browser plug-in supports smooth streaming.

3.2.1 Why MP4?

Microsoft has proposed several reasons for their migration from their Advanced Systems Format (ASF) [40] to MP4. Four of these reasons are:

Lightweight	the MP4 container format is lightweight (i.e. less overhead than the "asf" format)
Simple Parsing	parsing an MP4 container in .Net code is easy
H.264 CODEC support	the MP4 container supports the standard H.264 CODEC
Fragmentation	an MP4 container has native support for payload fragmentation

3.2.2 Disk File Format

Smooth Streaming defines a disk file format for a contiguous file on the disk (see figure 3.2). The basic unit of a container is referred to as a box. Each box may contain both data and metadata.

This disk file format contains Movie Meta Data (moov) - basically file-level metadata. The fragment section contains the payload. We have shown only two fragments in the figure; however, there could be more fragments depending upon the file's size. Each fragment section consists of two parts: a Movie Fragment (moof) and media data (mdat). The moof section carries more accurate fragment level metadata and the media is contained in the mdat section. Random access

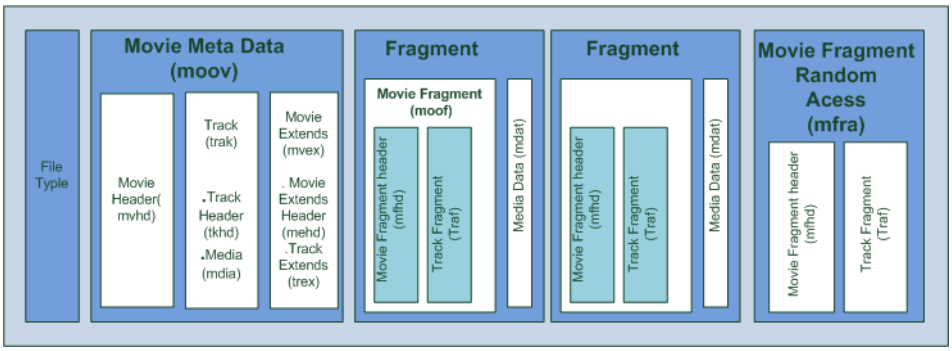


Figure 3.2: Disk File Format, adapted from [8]

and accurate seeking within a file is provided by the Movie Fragment Random Access (mfra) information.

3.2.3 Wire File Format

The wire file format is a subset of the disk file format, because all fragments are internally organized as a MP4 file. If a client requests a video time slice from the server (i.e., video from a starting time to and ending time), then the server seeks to the appropriate fragment file from within the MP4 file and transports this fragment file to the client. Figure 3.3 shows the wire file format.

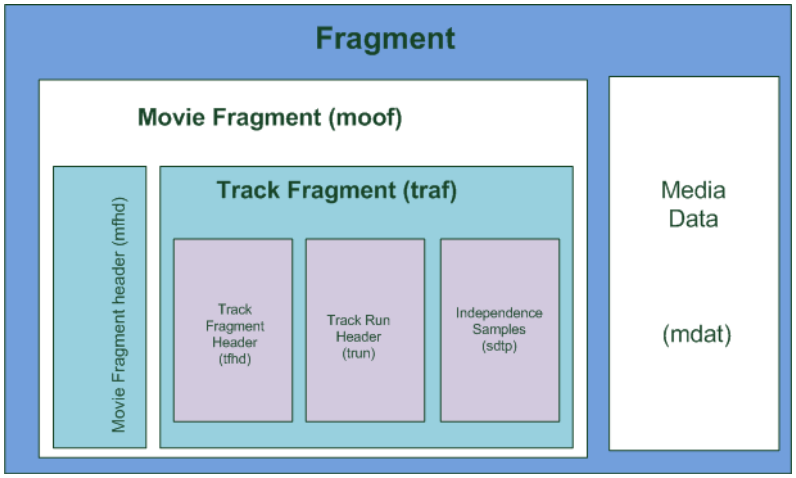


Figure 3.3: Wire file format, adapted from [8]

3.2.4 Media Assets

MP4 files	In order to differentiate from the traditional MP4 files , Smooth Streaming uses two new file extensions ".isma" and ".ismv". A file with the first extension contains only audio. A file with the second extension contains video and optionally can contain audio.)
Server manifest file (*.ism)	This file describes the relationship between the media tracks, the available bit rates, and the files on disk.
Client Manifest file (*.ismc)	This file describes the availability of streams to the client. It also describes what CODECs are used, the encoded bit rates, video resolution, and other information.

3.2.5 Smooth Streaming Playback

In Smooth Streaming, a player (client) requests a Client Manifest file from the server. Based on this information the client can initialize the decoder at runtime and build a media playout pipeline for playback. When the server receives the client's request, it examines the relevant server manifest files and maps the requested file to a MP4 file (e.g., a file with the extension of either .isma or .ismv) on disk. Next it reads the MP4 file and based on its Track Fragment (tfra) index box, it finds the exact fragment (containing the moof and mdat) corresponding to the client request. After that, it extracts the fragment file and sends it to the client. The fragment that is sent can be cached in Amazon's CloudFront for rapid delivery to other clients that request this same fragment (i.e., requesting the same URL). For example:

http://Serveraddress/server.ism/QualityLevels(bitrate)/Fragments (video=fragment number)

The above URL is used to request a fragment with specific values of bit rate and fragment number. The bit rate and fragment numbers are determined from the client manifest file.

3.3 Advertisement Insertion

Digital Program Insertion (DPI) allows the content distributor to insert digitally generated advertisements or short programs into the distributed program [41]. An SCTE 35 message is used for control signaling and an SCTE 30 [42] message is used for communication between splicer and content distributor. Most of the industry oriented advertisement insertion solutions are based on SCTE 35.

Cisco's advance advertising solution [43] supports SCTE 35 and uses a digital content manager for splicing. It also supports video insertion for video, before or during playback. Figure 3.4 shows Cisco's advanced advertising solution. An SCTE 35 digital cue tones is identified for the advertisement insertion opportunity and then, splicing digital content manager uses SCTE 30 message to collect the advertisement and splices accordingly, and send it to the end user.

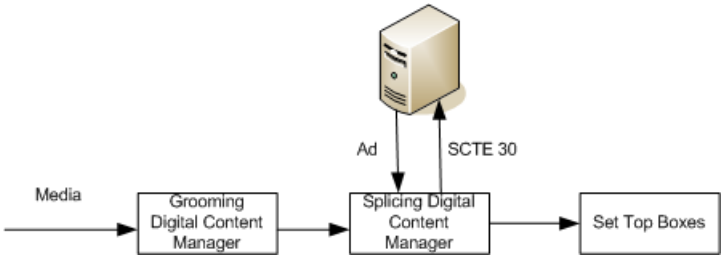


Figure 3.4: Cisco's advertising solution, adapted from [43]

There are some DPI monitor tools [44] [45] [46] that detect SCTE 35 digital cue tones. Alcatel Lucent's advertisement insertion [47] is also based on SCTE 35. Packet Vision has created a "Ad Marker Insertion System" that also supports SCTE 35 ad markers in the program stream for precise timing of advertisement insertion[48].

Innovid's platform [49] provides advertisement insertion in suitable areas of a video. It detects suitable advertisement insertion areas within the video, such as table or empty wall in the background. After mapping the ad space onto the video content, their ad server selects a specific advertisement for this mapped space. Advertisements are dynamically served to the mapped space at viewing time. These advertisements allow two-way interaction with between the user and the advertisement.

Chapter 4

Design and Implementation of a Prototype Streaming System with Dynamic Insertion of Advertisements

4.1 Design Overview

The proposed solution for a HTTP Streaming Server with Dynamic Advertisement was implemented using Java technologies, Python, a MPEG-2 TS container, and FFMPEG [50]. Details of each of these will be presented in this chapter. A laptop computer, an iPhone, a set top box (STB), and a Sony PlayStation are used as clients for testing our implementation. The overall system architecture is illustrated in figure 4.1 and it shows several modules communicating with each other. Initially, files are transcoded using a transcoder module and then segmented into chunks. Finally, as noted previously they are stored in three different places in the network: Amazon CloudFront, Operators Storage, and Internal Storage. The following steps are performed in conjunction with a client's request for media.

1. Client sends a resource request to the streaming server.
2. Streaming server fetches media based upon the Uniform Resource Locator (URL) [51] and requests the media's location from the node (i.e. tracker), who keeps all the media information along with the advertisement timing information.
3. Streaming server retrieves the media's location from the node.

4. Streaming server requests the resource chunks from their respective locations.
5. Streaming server fetches the contents received from the storage server and synchronizes their clock information.
6. Finally, the server combines the previously transcoded chunks together with the advertisement at run-time and streams the results to the client.

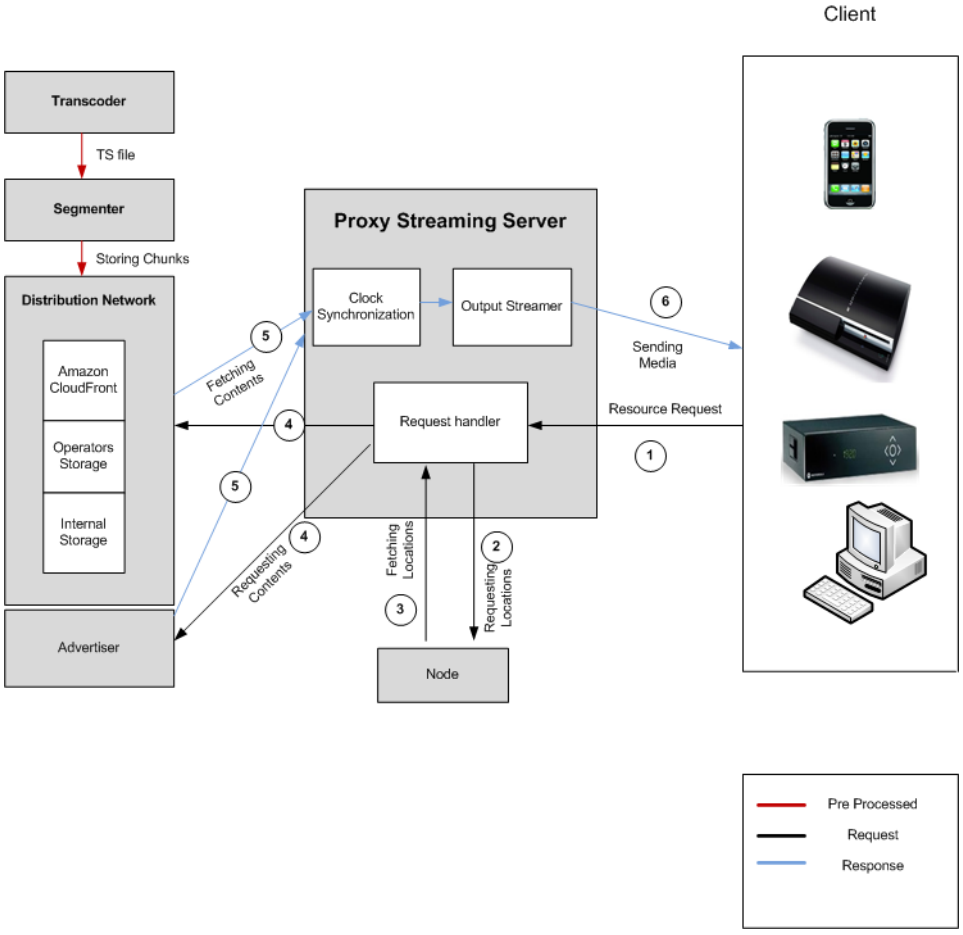


Figure 4.1: Overall Architecture

The overall message flow for requesting and providing the media is shown in figure 4.2. The following subsections describe how to choose the appropriate container and give details of the modules shown in figure 4.1.

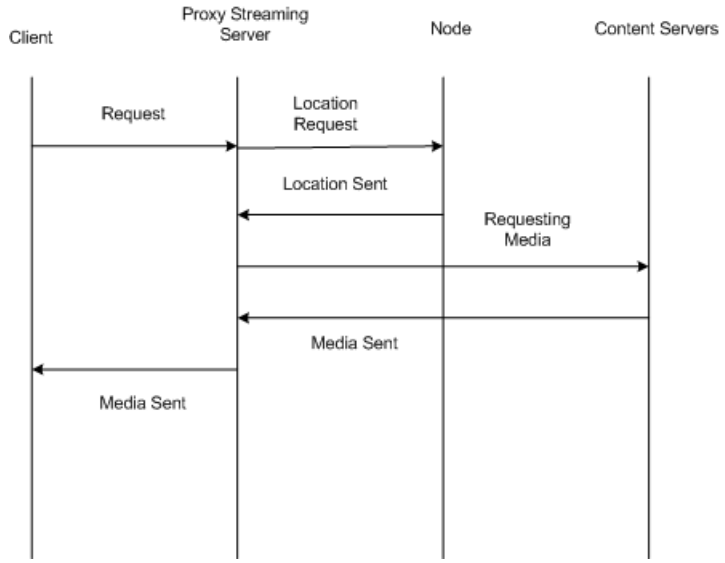


Figure 4.2: Message Flow

4.1.1 Choosing an Appropriate Container

One of the key research issues that needed to be addressed by this implementation was to select an appropriate container. The two most widely used containers for chunk based adaptive streaming are fragmented MP4 (FMP4) and MPEG2-TS. We explored both alternatives. Detailed information regarding streaming with FMP4 can be found in appendix B. MP4Split [52] was used for generating the FMP4 file and MP4Explorer [53] was used to analyze the file structure. Unfortunately, fragmented MP4 is only supported by Microsoft's Silverlight player. As a result our implementation uses MPEG 2-TS as the container because all the devices (laptop, Apple iPhone, STB, and Sony PlayStation) selected for our testing can play an MPEG 2-TS file.

4.1.2 Transcoding

The transcoding encodes the video using a H.264 CODEC and the audio using the AAC/MP3 CODEC and places the output in a MPEG-2 TS container (shown as a TS File in Figure 4.3). The media server stores the resulting encoded chunks.

4.1.3 Segmentation

Segmentation is the process of dividing the content into several parts. Segmentation takes an input video stream and produces several fixed length chunks. A segmenter developed in Python has been used to segment the video into several fixed length chunks. Figure 4.3 illustrates the combined Transcoding and Segmentation processes. In our implementation, the segmentation process provides the chunk based HTTP streaming.

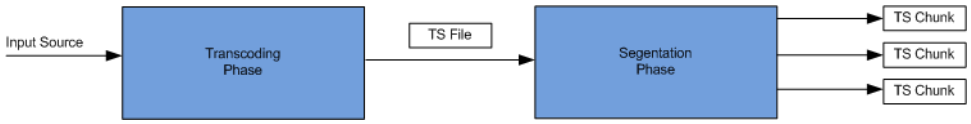


Figure 4.3: Transcoding and Segmentation

4.1.4 Distribution Network

After the transcoding and segmentation phase, segmented chunks are stored in three different places in the network. As noted previously the segmented chunks are stored in: Internal Storage, Amazon CloudFront, and Operators Storage.

4.1.5 Client Devices

The list of client devices that have been used in the project are summarized in Table 4.1 . These client devices are responsible for initiating a resource request to the proxy server.

Table 4.1: Client Hardware

Client Devices	Description
Laptop computer	An HP Pavilion dv7 laptop was used for both development and testing of a client. The configuration of this laptop was as follows: • Processor: Intel core 2 Duo P8400 2.26 GHz • RAM: 2GB • Disk Space: 120GB
Apple iPhone 3G	An Apple iPhone 3G used for testing as client was configured as follows: • OS version: 3.0 • Firmware Version: 04.26.08 • Disk Space : 8 GB
Motorola Kreatel TV STB	VIP 1970-9T used for testing as client was configured as follows: • Architecture - MIPS • CPU Speed - 266 MHZ • Main Memory - 128 MB • Disk Space - 160GB
Sony PlayStation 3	Sony PlayStation 3 used for testing as a client was configured as follows: • Processor - IBM Cell 3.2 GHz • Disk Space - 60 GB

4.1.6 Proxy Streaming Server

This project implemented a streaming server that receives a media request from a client, then fetches the requested media contents and combines this media with an advertisement before providing the combined results to the client. This streaming proxy server acts as both a server and a client. It acts as a client to fetch content from the media server, but it acts as a server to the end-user's client. A proxy server acts as a mediator between clients and content servers to accept client requests for content from other web servers [54]. In general, the client connects to a proxy server to request a resource. We developed our own proxy streaming server, as most of the client devices used for testing cannot handle an HTTP REDIRECT request. (Table 4.2 summarizes the language and libraries used for developing our proxy streaming server.)

Table 4.2: Programming Languages and Application Server

Client Devices	Description
JAVA	J2EE (servlet), ApacheHTTPClient Library [55] used by the streaming server to handle HTTP GET requests from clients.
Python	Python is used for parsing the MPEG2-TS header and synchronizing the clock information.
GlassFish	Application server used for Proxy.

4.1.6.1 Request Handler

When a client sends a HTTP GET request to the proxy server, the proxy parses the Uniform Resource Locator (URL) [51] to find out which asset is requested. Figure 4.4 illustrates this request handling. Client requests the media using the proxy server's URL and media asset's ID (http://proxyserverurl:8080/mediaassetid). After that, the proxy fetches the requested media using this URL and retrieves the media asset ID and requests the media locations from the node for asset and sends HTTP GET requests to the respective locations in the network.

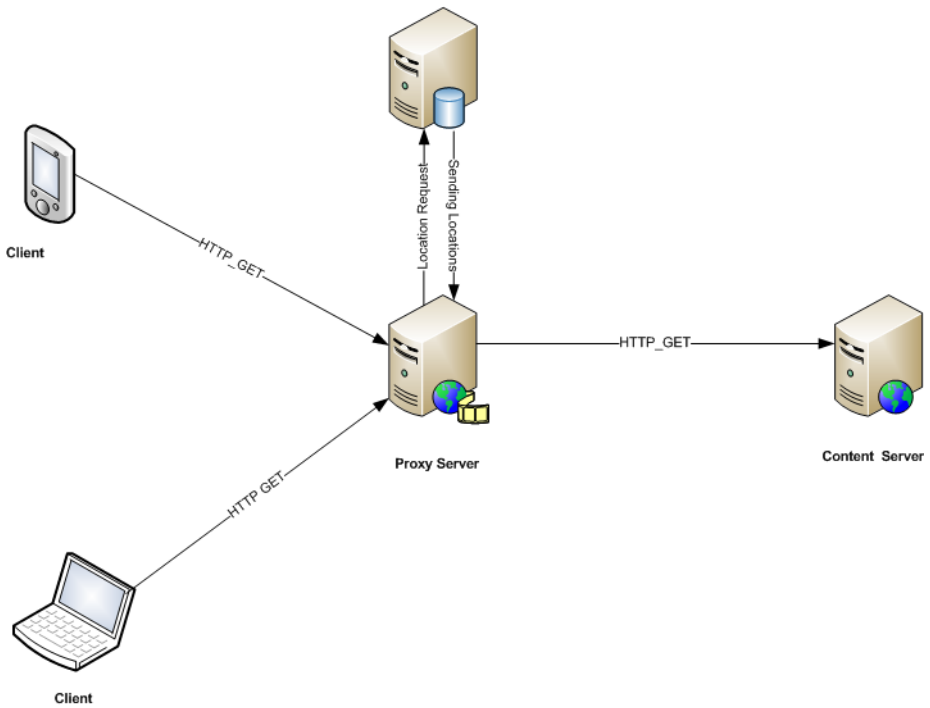


Figure 4.4: Request Handling

4.1.6.2 Clock Synchronization

Advertisement insertion in the TS file for continuous streaming requires the synchronization of the clock, specifically the PCR, PTS or DTS value. However, this requires parsing the TS header, PAT and PMT tables, and parsing PES header. We started our research by setting the Discontinuity Indicator (DI) field true in the TS header for advertisement that needs to be inserted in the stream and after that we tested the files with different client devices. All of the devices except for the iPhone were able to play the stream. In order to add an advertisement in a movie, iPhone requires modification of the clock information to let the device know that the advertisement is a continuation of the same movie.

4.1.6.3 Setting the Discontinuity Indicator

To set the Discontinuity Indicator (DI) in the advertisement chunk we need to parse first 4 bytes of TS header and check if there is any adaptation field present. If so, then we parse the adaptation field and set the DI. Listing 4.1 shows the algorithm of setting the DI.

Listing 4.1: Setting the Discontinuity Indicator

```
TS_Packet_Length = 188
TS_Header_Size = 4
Take TS_Packet_Length bytes
Parse TS_header_Size from TS_Packet_Length
If Adaptation_Field_Control==2 || Adaptation_Field_Control==3
  Parse Adaptation Field
  Set the Discontinuity_Indicator
```

4.1.6.4 Changing the Program Clock Reference

To make a continuous streaming flow, the PCR value of the advertisement chunk should continue from the Last PCR value and PCRs for the following chunks of the movie should be modified to follow the last PCR of the advertisement. Listing 4.2 shows the algorithm for changing the timestamps.

Listing 4.2: Changing Program Clock Reference

```
TS_Packet_Length = 188
TS_Header_Size = 4
Take TS_Packet_Length bytes
// To get PID and Adaptation Field Control
Parse TS_header_Size from TS_Packet_Length
Parse PAT and PMT table  to get PCR_PID
If PID == PCR_PID    // PID
  If Adaptation Field Control ==2 || Adaptation Field Control ==3
    Parse Adaptation Field
   If PCR Flag == 1
      Read PCR
    Check PCR with the Previous PCR
    If  current PCR != Previous PCR
      //To check the continuity
      Change the PCR
      Store the changed PCR
        //for comparing with next
    else
        Store the PCR
       //for comparing with next
```

4.1.6.5 Changing Time Stamp

The Presentation Time Stamp (PTS) or Decoding Time Stamp (DTS) should also be changed for the inserted advertisement and should follow the last PTS/DTS value of the movie where the advertisement is going to be inserted and the PTS/DTS of all the remainder of the movie should be changed to follow the PTS/DTS of the last advertisement chunk. Listing 4.3 shows the algorithm of changing the timestamps.

Listing 4.3: Changing the Time Stamp

```
TS_Packet_Length = 188
TS_Header_Size = 4
Take TS_Packet_Length bytes
// To get PID and Adaptation Field
Parse TS_header_Size from TS_Packet_Length
Parse PAT and PMT table to get Elementary_Stream_IDs
If PID in Elementary_Stream_IDs
  If adaptation field is present
     H = TS_header_Size + Adaptation_Field_Length
     Parse PES header after H bytes
    If PTS or PTS_DTS flag is present
      Read PTS/DTS
     If  current PTS/DTS != previous PTS/DTS
       Change the PTS/DTS
      Store the PTS/DTS
         //for comparing with the next PTS/DTS
     else
      Store the PTS/DTS
      //for comparing with the next PTS/DTS
  else
    Parse PES header after TS_Header_Size
    If PTS or PTS_DTS flag is present
     Read PTS/DTS
     If  current PTS/DTS != previous PTS/DTS
      Change the PTS/DTS
      Store the PTS/DTS
       //for comparing with the next PTS/DTS
     else
      Store the PTS/DTS
        //for comparing with the next PTS/DTS
```

4.1.6.6 Output Streamer

To stream the file as a single HTTP resource to the client, the proxy streaming server splices the content together after performing the clock synchronization. The proxy streaming server reads bytes continuously and modifies the header information and places the output bytes in a output pipe after one another. Therefore, the client on the other side of the output pipe experiences the output as a single HTTP resource. Figure 4.5 shows the splicing process of the proxy server.

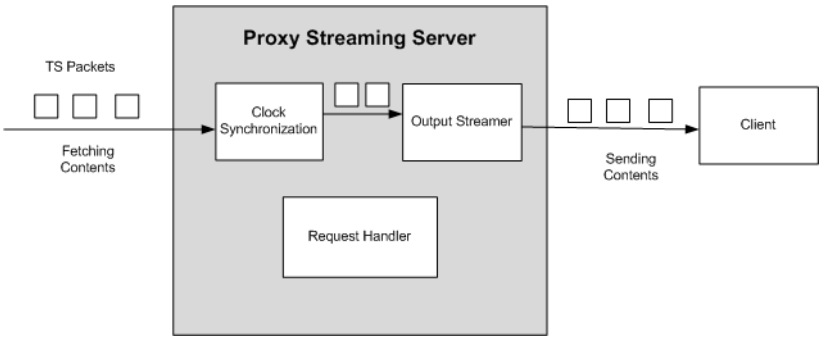


Figure 4.5: Output Streamer

4.2 Advantages of Dynamic Advertisement Insertion

The advantages introduced by the proposed dynamic advertisement insertion solution are described in the following paragraphs.

4.2.1 Reduce Storage Cost

Reducing the storage cost at the server is the main advantage introduced by the proposed solution. This solution can combine the same video chunks with several different advertisements, thus lowering the storage costs at the server side (in comparison to having to create a separate complete file for each combination of content and advertisements). For example, in a traditional advertisement insertion system, if there are five advertisements then creating a version of the content together with each of these five advertisements takes four times more space than a single copy of a movie. However, using adaptive chunk based HTTP streaming with the implemented dynamic ad insertion solution; the storage cost at the server's side requires storage only of the movie and each of the advertisements.

4.2.2 Runtime Decision for Advertisement Insertion

The implemented solution has the advantage of inserting the advertisement at runtime. Since the advertisement is inserted into the content at runtime, providers have the flexibility of changing the advertisement or altering the location (with respect to the relative time that the advertisement is presented in the media stream) of an advertisement at runtime.

4.2.3 Personalized Advertisement Insertion

Dynamic advertisement insertion allows personalized advertisement insertion. Using this context information, it is possible to select the best matching

advertisement for a specific client for insertion at runtime. For example, a Swedish-speaking user might prefer to watch a Swedish advertisement rather than an advertisement in German.

4.2.4 Advertisement Insertion based on Geographical and IP Topological Location

Advertisement insertion based on geographic location and/or IP topological location has been widely exploited in advertisement based revenue schemes. The best matching advertisements can be selected based on the user's geographic location. For example, a user in Sweden might be presented with a advertisement for a product that is actually available in Sweden rather than a product that is only available in Japan. Alternatively, it is possible to insert an advertisement based on the user's location within the network (for example, presenting an advertisement that is made available via the local network operator - this might even be a very local network operator - such as the owner of a cafe that provides Wi-Fi access to its customers).

4.3 Disadvantages of the proposed solution

The implemented solution does not scale well since it has to do advertisement insertion for the sum of all of the clients that are viewing streaming content with advertisements. This should be contrasted with the alternative solution of using client based advertisement insertion.

Chapter 5

System Analysis

This section describes the analysis of our implemented system. The system has been analyzed by performing validity checking of the resulting content plus advertisement (as a TS stream), by measuring the transaction time and download response time when using a proxy streaming server. Details of the system testing with vendor specific devices can be found in appendix D.

5.1 Validity checking of a TS file

To achieve continuous flow of a TS stream, we have modified the clock information of TS streams. To check whether the modified TS file is valid or not we have used an MPEG-TS analyzer [56] to show the header information. The analyzer is unable to read the TS packet if the packet is corrupted. Figure 5.1 shows a screenshot from the analyzer for a TS packet.

Figure 5.2 shows the header information of first few TS packet to illustrate the header information after the modification (including the PCR, PTS and DTS value). In addition to that, movie player is able to play the modified file.

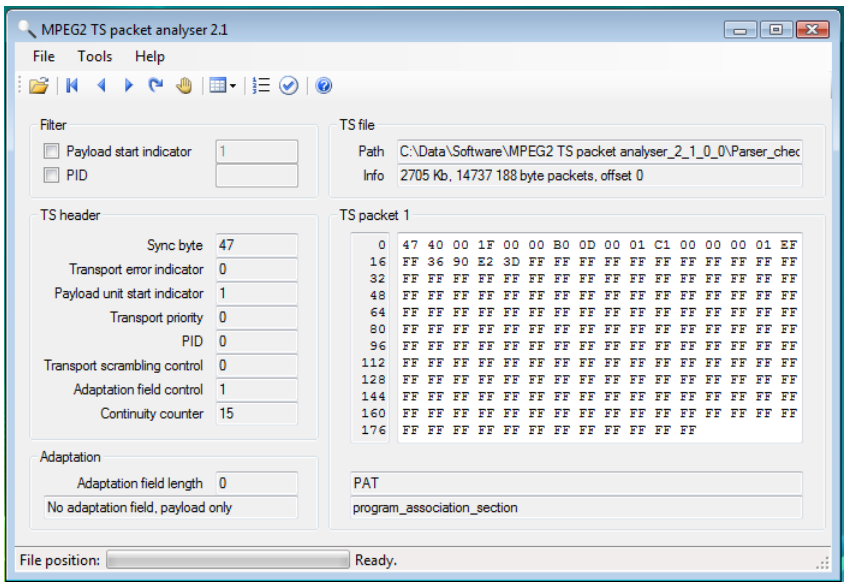


Figure 5.1: TS packet analyzer

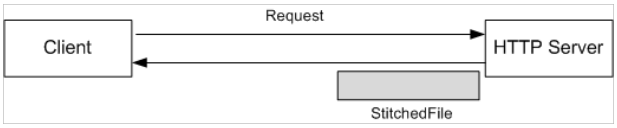
packet	PID	sync	transport_error	payload_start	transport_priority	scrambling_control	adaptation_control	continuity_count	PID_info	Adap.length	discontinuity_indicator	Adaptation.PCR	PES.PTS	PES.DTS
1	0	71	0	1	0	0	1	15	PAT	0	FALSE	0	0	0
2	4095	71	0	1	0	0	1	15		0	FALSE	0	0	0
3	256	71	0	1	0	0	1	4		0	FALSE	0	5702412	0
4	256	71	0	0	0	0	1	5		0	FALSE	0	0	0
5	256	71	0	0	0	0	1	6		0	FALSE	0	0	0
6	256	71	0	0	0	0	1	7		0	FALSE	0	0	0
7	256	71	0	0	0	0	1	8		0	FALSE	0	0	0
8	256	71	0	0	0	0	1	9		0	FALSE	0	0	0
9	256	71	0	0	0	0	1	10		0	FALSE	0	0	0
10	256	71	0	0	0	0	1	11		0	FALSE	0	0	0
11	256	71	0	0	0	0	1	12		0	FALSE	0	0	0
12	256	71	0	0	0	0	3	13		8	FALSE	1269740400	0	0
13	256	71	0	0	0	0	1	14		0	FALSE	0	0	0
14	256	71	0	0	0	0	1	15		0	FALSE	0	0	0
15	256	71	0	0	0	0	1	0		0	FALSE	0	0	0
16	256	71	0	0	0	0	1	1		0	FALSE	0	0	0
17	256	71	0	0	0	0	1	2		0	FALSE	0	0	0
18	256	71	0	0	0	0	1	3		0	FALSE	0	0	0
19	256	71	0	0	0	0	1	4		0	FALSE	0	0	0
20	256	71	0	0	0	0	1	5		0	FALSE	0	0	0
21	256	71	0	0	0	0	1	6		0	FALSE	0	0	0
22	256	71	0	0	0	0	1	7		0	FALSE	0	0	0
23	256	71	0	0	0	0	1	8		0	FALSE	0	0	0
24	256	71	0	0	0	0	1	9		0	FALSE	0	0	0
25	256	71	0	0	0	0	1	10		0	FALSE	0	0	0
26	256	71	0	0	0	0	3	11		8	FALSE	1270248600	0	0
27	256	71	0	0	0	0	1	12		0	FALSE	0	0	0
28	256	71	0	0	0	0	1	13		0	FALSE	0	0	0
29	256	71	0	0	0	0	1	14		0	FALSE	0	0	0
30	256	71	0	0	0	0	1	15		0	FALSE	0	0	0
31	256	71	0	0	0	0	1	0		0	FALSE	0	0	0
32	256	71	0	0	0	0	1	1		0	FALSE	0	0	0
33	256	71	0	0	0	0	1	2		0	FALSE	0	0	0
34	256	71	0	0	0	0	1	3		0	FALSE	0	0	0
35	256	71	0	0	0	0	1	4		0	FALSE	0	0	0
36	256	71	0	0	0	0	1	5		0	FALSE	0	0	0
37	256	71	0	0	0	0	1	6		0	FALSE	0	0	0
38	256	71	0	0	0	0	1	7		0	FALSE	0	0	0
39	256	71	0	0	0	0	1	8		0	FALSE	0	0	0
40	256	71	0	0	0	0	3	9		8	FALSE	1270756800	0	0

Figure 5.2: TS packet information

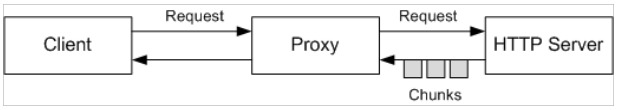
5.2 Measuring Transaction Time and Download Response Time using Proxy Streaming Server

This thesis project analyzed the transaction time and download response time from when the clients requests content through the proxy streaming server until the content is provided to the client. Three scenarios have been considered to check whether the proxy is able to deliver the video within an appropriate delay bound compared to two other scenarios where clients request the files directly from the server. Figures 5.3a , 5.3b, and 5.3c illustrate the three scenarios:

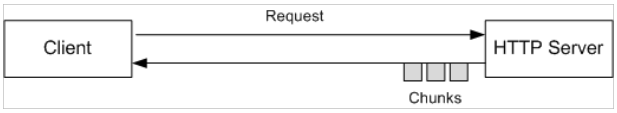
1. Client requests one stitched file from the content server directly;
2. Client requests several chunks from the content server through the proxy streaming server; and
3. Client requests several chunks directly from the content server.



(a) Client requests one stitched file



(b) Client requests several chunks through proxy



(c) Client requests multiple chunks

Figure 5.3: (a) Client requests one stitched file from the content server directly; (b) Client requests several chunks from the content server through the proxy streaming server; and (c) Client requests several chunks directly from the content server

The following subsections give details of the test environment, procedures, data collection, and analysis.

5.2.1 Test Environment

To perform the test we have used an HTTP Server for storing the contents, a proxy streaming server, and a client running Microsoft's Winodows Vista. All of these computers are connected to an unmanaged 1Gbit/s Ethernet Switch. The specifications of the machine used for our testing are:

Proxy Streaming Server	Proxy Streaming Server used for analysis was configured as follows: CPU: Intel Pentium D 3.2 GHz RAM: 2048 MB OS: Ubuntu 9.10 (Kernel: 2.6.31) Application Server: Glassfish v2.1 NIC: 1 Gbit/s
HTTP Server	HTTP Server used for analysis was configured as follows: CPU: Intel Atmon 360 1.6 GHz RAM: 2048MB OS: Windows7 HTTP Server: Apache2 NIC: 1 Gbit/s
Client Machine	Client machine used for analysis was configured as follows: Processor: Intel core 2 Duo P8400 2.26 GHz RAM: 2 GB Disk Space: 120GB OS: Windows Vista NIC: 1 Gbit/s
Netgear GS 108	1 Gbit/s Ethernet switch

We transcoded the files with the following command for testing:

Listing 5.1: Transcoding command

```
ffmpeg -i inputfile -f mpegts -acodec libmp3lame -ar 48000 -ab 64k
    -s 320x240 -vcodec libx264 -b 1000K -flags +loop -cmp +chroma -
    partitions +parti4x4+partp8x8+partb8x8 -subq 5 -trellis 1 -refs
    1 -coder 0 -me_range 16 -keyint_min 25 -sc_threshold 40 -
    i_qfactor 0.71 -bt 200k -maxrate 1000K -bufsize 1000K -rc_eq '
    blurCplx^(1-qComp)' -qcomp 0.6 -qmin 10 -qmax 51 -qdiff 4 -level
     30 -aspect 320:240 -g 30 -async 2 output.ts
```

5.2.2 Test Procedure

We developed a Java Client that can make concurrent requests. Initially, we started our test with 1 request, then 2 requests, and increased the number of requests in increments of 2 until reaching 20 concurrent requests. Then 20 tests were run for each different number of concurrent requests. Using our java client, we calculated the transaction time and download response time for the requests. We performed the test for all the three scenarios shown in figures 5.3a , 5.3b,

and 5.3c. Using the same test environment and test procedure, we measured the download transaction time for 1 to 46 concurrent requests for a file size of 10 MB, details of these measurements can be found in appendix E.

5.2.3 Transaction Time

5.2.3.1 Client requests one stitched file from the content server directly

Data Collected

We calculated the average value and standard deviation for the tests for each different number of concurrent requests. Note that the systems were all idle except for the processing of these requests. No attempt was made to terminate background tasks or otherwise adapt the systems from their default configurations. Table 5.1 summarizes the results.

Table 5.1: Average transaction time and standard deviation value - Client requesting one stitched file directly from the server

Concurrent requests	Average transaction time (ms)	Standard Deviation
1	2563.05	2095.22
2	2490.43	2123.94
4	4374.98	1334.07
6	8252.58	1733.06
8	11224.92	1790.56
10	15487.10	1748.62
12	19134.77	2136.12
14	23992.81	2427.93
16	26563.13	2665.90
18	30867.27	2725.81
20	35304.47	2956.99

5.2.3.2 Client requests several chunks from the content server through the proxy streaming server

Data Collected

Table 5.2 summarizes the average value and standard deviation for the tests for each different number of concurrent requests.

Table 5.2: Average transaction time and standard deviation value - client requests several chunks through proxy

Concurrent requests	Average transaction time (ms)	Standard Deviation
1	2096.65	83.66
2	2757.53	79.65
4	5385.48	183.64
6	6993.93	790.27
8	8197.56	1965.78
10	9935.37	3294.71
12	10847.25	3756.51
14	12163.55	4699.40
16	13654.42	5638.94
18	14818.98	6375.36
20	16650.07	7404.54

5.2.3.3 Client requests several chunks directly from the content server

Data Collected

We calculated the average value and standard deviation for the tests for each different number of concurrent requests. Table 5.3 summarizes the results.

Table 5.3: Average transaction time and standard deviation value - Client requests several chunks directly from the content server

Concurrent requests	Average transaction time (ms)	Standard Deviation
1	2391.55	1746.96
2	2614.28	1868.99
4	4426.06	1272.70
6	7345.19	1225.63
8	10572.97	1354.57
10	13588.53	1402.28
12	16909.55	1665.63
14	19389.44	1350.56
16	21927.54	1641.33
18	24872.35	1968.74
20	28189.21	1881.89

Data Analysis

Figures 5.4, 5.5, and 5.6 show the graph for a client requesting one stitched file directly from the HTTP server, a client requesting multiple chunks directly, and a client requesting multiple chunks through the proxy. Here the x axis represents the number of concurrent requests and the y axis represents the average values of the transaction time. We performed a regression analysis and found that all the graphs follow a polynomial function. But, for more than 4 concurrent requests, transaction time increases linearly.

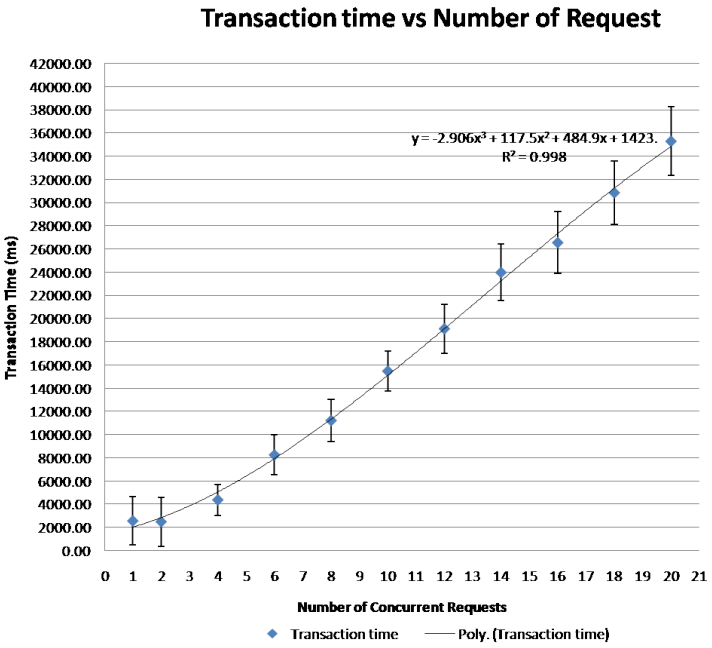


Figure 5.4: Transaction time vs number of concurrent requests - Client requesting one stitched file directly from the server

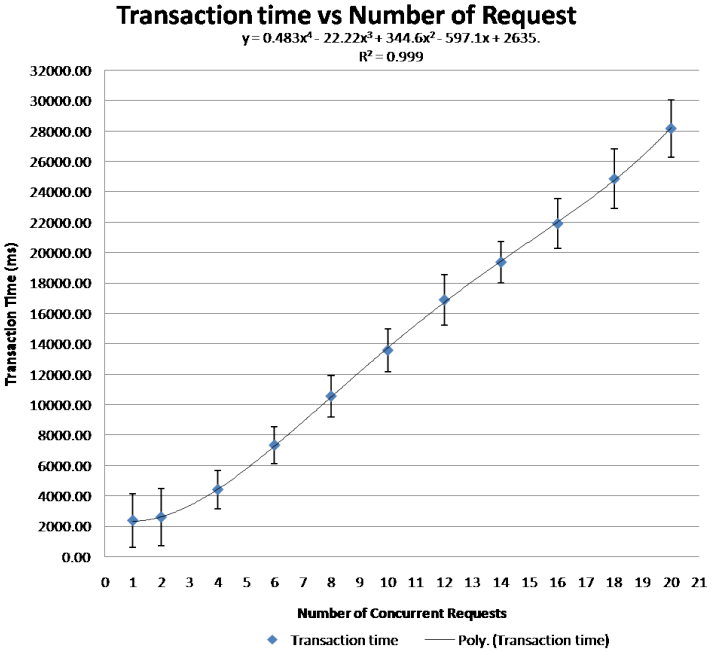


Figure 5.5: Transaction time vs number of concurrent requests - client requests several chunks directly from the content server

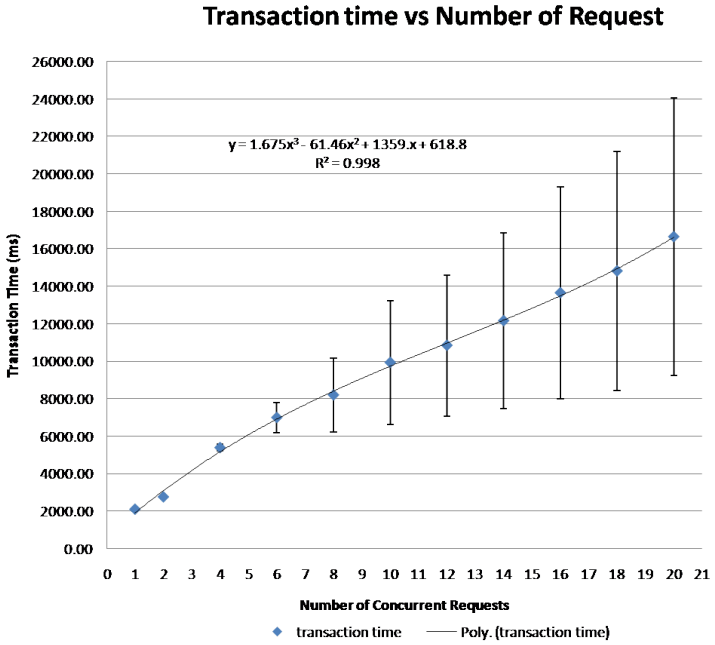


Figure 5.6: Transaction time vs number of concurrent requests - client requests several chunks through proxy

We have checked the apache configuration to see why download time is increasing linearly for more than 4 concurrent requests and found that apache server was not optimized. Server needs to create extra thread to handle more than 5 concurrent requests. This is why we can see that there is a linear growth in the download time for more than four councurrent requests. Apache HTTP server was configured with the following information:

- Start servers - 5
- MinSpareServer - 5
- MaxSpareServer - 10
- MaxClient - 150
- MaxRequestperchild - 0

We have used iperf to check the bandwidth for client to content server, client to the proxy streaming server, and content server to the proxy streaming server and found the bandwidth is 900 Mb/s. Based on the calculated bandwidth, the time for a single client to get a file of 10 MB is 1200 ms if they all share the same bandwidth. Figure 5.7 illustrates the comparison graph for all the three scenarios for more than 4 concurrent requests.

We can see from the graph that transaction time is increasing linearly for more than 4 concurrent requests for the three scenarios and transaction time for client requesting through proxy is less compared to the direct delivery of the file.

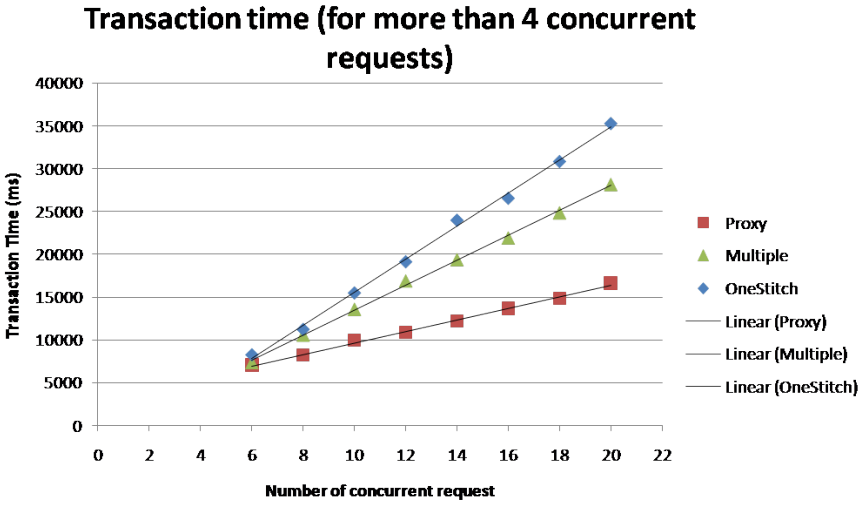


Figure 5.7: Comparison graph

Based on the measured bandwidth, we calculated excess time required for all the scenarios and are shown in table 5.4.

Table 5.4: Excess transaction time for sending the file

Concurrent requests	**Time for sending the file (ms)**	**Excess time - client requesting one stitched file**	**Excess time - client requesting through proxy**	**Excess time - client requesting multiple chunks**
1	1200	1363.05	896.65	1191.55
2	2400	90.43	357.53	214.28
4	4800	−425.02	585.47	−373.94
6	7200	1052.58	−206.08	145.19
8	9600	1624.92	−1402.44	972.97
10	12000	3487.10	−2064.63	1588.53
12	14400	4734.77	−3552.75	2509.55
14	16800	7192.81	−4636.45	2589.44
16	19200	7363.13	−5545.58	2727.54
18	21600	9267.27	−6781.03	3272.35
20	24000	11304.47	−7349.94	4189.21

5.2.4 Response Time

5.2.4.1 Client requests one stitched file from the content server directly

Data Collected

Table 5.5 summarizes the average value and standard deviation for the tests for each different number of concurrent requests.

Table 5.5: Average response time and standard deviation value - Client requests one stitched file directly from the content server

Concurrent requests	Average response time (ms)	Standard Deviation
1	43.55	13.90
2	45.53	4.21
4	66.46	49.39
6	76.93	44.78
8	107.78	77.85
10	132.12	88.77
12	166.53	108.61
14	187.14	123.67
16	226.56	141.91
18	237.03	142.90
20	252.15	152.38

5.2.4.2 Client requests several chunks directly from the content server

Data Collected

We calculated the average response time and standard deviation for the tests for each different number of concurrent requests. Table 5.6 summarizes the results.

Table 5.6: Average response time and standard deviation value - Client requests several chunks directly from the content server

Concurrent requests	Average response time (ms)	Standard Deviation
1	46.55	45.54
2	42.43	4.38
4	62.81	37.94
6	73.62	36.53
8	125.81	85.94
10	148.87	88.83
12	187.75	111.61
14	178.12	95.01
16	216.66	113.46
18	292.59	151.29
20	304.85	168.92

5.2.4.3 Client requests several chunks from the content server through the proxy streaming server

Data Collected

Table 5.7 summarizes the average value and standard deviation for the tests for each different number of concurrent requests.

Table 5.7: Average response time and standard deviation value - Client requests several chunks through the proxy streaming server

Concurrent requests	Average response time (ms)	Standard Deviation
1	51.75	3.34
2	60.93	4.59
4	87.83	16.62
6	1183.22	2430.80
8	2573.19	3206.79
10	3412.15	3322.26
12	4985.94	4703.06
14	6106.91	5173.51
16	7429.64	6063.41
18	8783.19	6883.20
20	10099.87	7469.37

Data Analysis

Figures 5.8, 5.9, and 5.10 illustrate the graph for client requesting one stitched file directly from the HTTP server, where the x axis shows the the number of concurrent requests and the y axis represents the average values of the download response time.

It can be seen from the graph that for more concurrent request, data tends to be more deviated from the average value when a client requests through the proxy streaming server. We performed a regression analysis and found that the graph follows a polynomial function. However, for more than 4 concurrent requests download response time is increasing linearly because the configuration of HTTP server was not optimized and download response time is higher for more concurrent requests for client requesting through proxy streaming server compared to client requesting files directly from the server. This was expected because number of requests through the proxy is one greater than the number of requests directly to the server. And, proxy streaming server is performing the file operatation to read the URL of the media files. Our proxy has been deployed in an application server of the MPM project but developing the proxy directly using JAVA or C and optimizing the apache configuration could lower the response time.

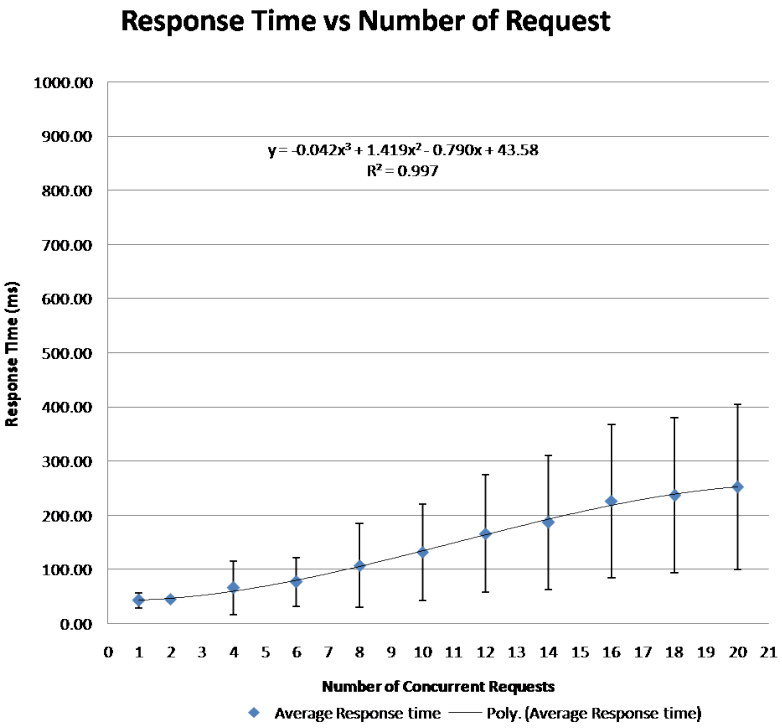


Figure 5.8: Response time vs number of concurrent requests - client requests one stitched file directly from the content server

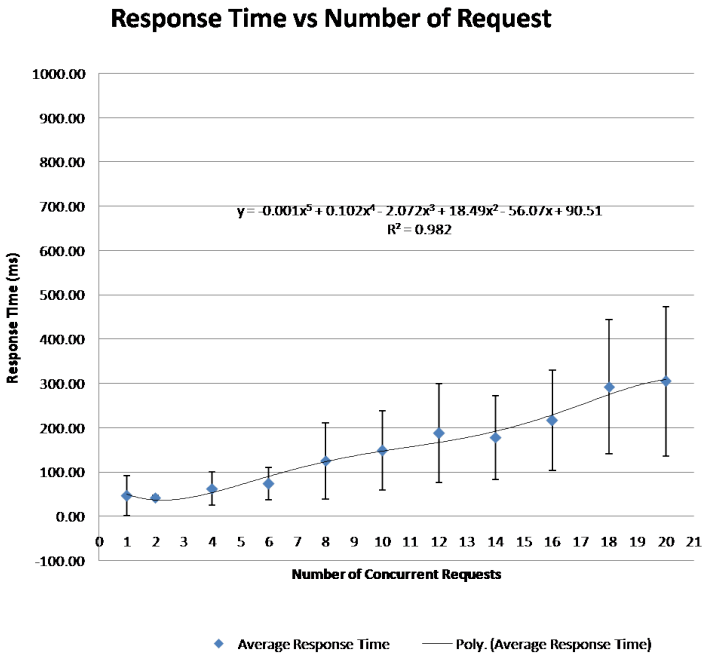


Figure 5.9: Response time vs number of concurrent request - client requests several chunks directly from the content server

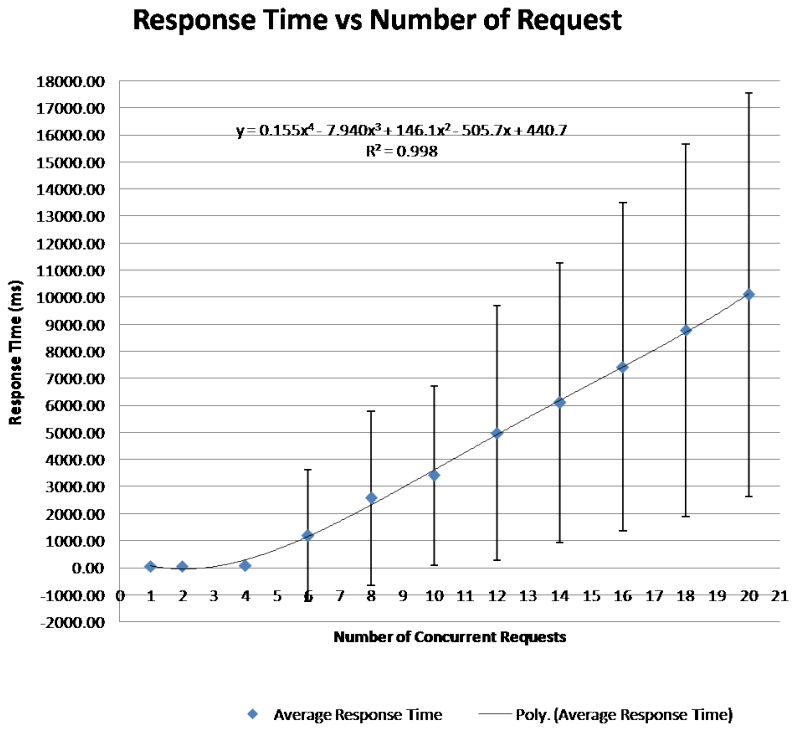


Figure 5.10: Response time vs number of concurrent requests - Client requests several chunks through proxy

Chapter 6

Conclusion

6.1 Summary of Work

The thesis project included extensive study to find out the best transport format for the media delivery and also studied two exisiting streaming solutions in order to select an appropriate solution. The thesis also documents the current state of the art (see section 3.1 and 3.2). The challenges of making a streaming video server that supports dynamic advertisement splicing were investigated and the research challenges were identified. A proxy streaming server was proposed and implemented. This proxy supports chunk based streaming and allows splicing of the advertisements into a media stream. The implemented solution was tested with a number of different devices.

6.2 Research Findings

The research questions stated in section 1.3 which were posted at the start of the thesis are summarized below with the findings from our work.

Question 1	To deliver the media chunks an appropriate container is required. Which is the most appropriate container?
Findings	This thesis recommends using MPEG-2 TS because it is supported by most of the different vendor's devices that we tested. Video splicing at the splicing boundary is quite flexible in MPEG TS.
Question 2	Can the solution be sufficiently portable that it will support different vendors' end-devices, such as Motorola's set top boxes (STBs), SONY's Play Station 3, and Apple's iPhone?
Findings	The proposed solution has been successfully tested with all three devices.
Question 3	How can we maintain the continuous flow of a stream including the advertisement? This means that it is very important to find out the proper splicing boundaries for advertisement insertion in order to maintain the stream's continuity.
Findings	To maintain the continuous flow of stream, we have modified the clock informations (specifically PCR, PTS and DTS).
Question 4	Can the solution be implemented successfully on a constrained server, while delivering media to the client within the appropriate delay bound?
Findings	The implemented system has been analyzed and we found that our implemented solution takes somewhat more transaction time when there are few concurrent request (i.e. 2 or 4). However, it takes less time for more concurrent request. We also found that configuration of our apache server was not optimized.

6.3 Future Work

The thesis work project has created platform for additional research work. Specifically the following future works are suggested:

- The whole solution was tested with a constant bitrate stream. The solution should be tested with variable bitrate streams in order to examine its ability to support adaptive streaming.
- More analysis parameters should be selected to evaluate the performance of the proxy server (i.e., changing the file size or transcoding parameters).

- While working with fragmented MP4, we found that it is possible to remove the fragment(moof) from the file using MP4split [52]. If we can split out all the fragments from the file and insert the advertisement fragment and then append the header for the file, then it would be possible to provide advertisement insertion support in a rather simple fashion. However, further study and investigation is required to implement this approach of inserting advertisements into for fragmented MP4 content.

- All of our processing was performed on plain text content, the measurements and evaluation should be repeated with encrypted content. Further development is required to make the system work with encrypted content.

Bibliography

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. IETF, Network Working Group, Request for Comments: 2616, June 1999. http://tools.ietf.org/html/rfc2616. [cited at p. 1]

[2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. IETF Network Working Group, Request for Comments: 3550, July 2003. http://www.ietf.org/rfc/rfc3550.txt. [cited at p. 1, 5]

[3] Alex Zambelli. A Brief History of Multi-Bitrate Streaming. Blog entry, December 17th 2008. Available from http://alexzambelli.com/blog/2008/12/17/a-brief-history-of-multi-bitrate-streaming/. [cited at p. 1]

[4] Using Advanced Advertising to Unlock Revenues. Visionary paper, Tandberg television (part of the Ericsson group), http://www.tandbergtv.com/uploads/documents/AdvancedAdvertising.pdf, 7 July 2009. Last accessed Sep, 2009. [cited at p. 2]

[5] M. Vorwerk, C. Curescu, H. Perkuhn, I. Ms, and R. Rembarz. Media Plane Management Reference Architecture. Technical Report EDD-09:000866, Ericsson, 27 April 2009. [cited at p. 2, 18, 19]

[6] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). IETF Network Working Group, Request for Comments: 2326, April 1998. http://www.ietf.org/rfc/rfc2326.txt. [cited at p. 5]

[7] Alex Zambelli. The Birth of Smooth Streaming. Blog entry, February 4th 2009. Available from http://alexzambelli.com/blog/2009/02/04/the-birth-of-smooth-streaming/. [cited at p. 5]

[8] IIS Smooth Streaming Technical Overview. Available From http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=03d22583-3ed6-44da-8464-b1b4b5ca7520;. [cited at p. 6, 15, 23]

[9] MPEG-2 Overview. http://www.fh-friedberg.de/fachbereiche/e2/telekom-labor/zinke/mk/mpeg2beg/beginnzi.htm;. Last Visited - September 2009. [cited at p. 7]

[10] Barry G. Haskell, Atui Puri, and Arun N Netravali. *An Introduction to MPEG-2*. New York. Springer-Verlag, second edition edition, 1996. ISBN-10: 0412084112 and ISBN-13: 978-0412084119. [cited at p. 7, 10]

[11] Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC. in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2003. JVTG050,2003. [cited at p. 8]

[12] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003. [cited at p. 8]

[13] I. Richardson. An Overview of H.264 Advanced Video Coding - White paper. http://www.videosurveillance.co.in/H.264.pdf. Last Visited - September 2009. [cited at p. 8]

[14] Marta Karczewicz and Ragip Kureren. The SP and SI frames Design for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):637–644, July 2003. [cited at p. 8]

[15] Stephan Wenger. H.264/AVC over IP. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):645–656, July 2003. [cited at p. 8]

[16] Ramin Soheili. MPEG-4 Part 10 - White paper. http://www.flexspeech.com/pdf/mpeg_4_part_10_white_paper_1.pdf. Last Visited-October 2009. [cited at p. 8]

[17] Henrique Malvar, Antti Hallapuro, Marta Karczewicz, and Louis Kerofsky. Low-Complexity Transform and Quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):598–603, July 2003. [cited at p. 9]

[18] MPEG-2 Transport Stream. http://en.wikipedia.org/wiki/MPEG_transport_stream. Last Accessed -September 2009. [cited at p. 10]

[19] Jerker Bjrkqvist. Digital Television Lecture Slide. Abo Akademi University, http://users.abo.fi/~jbjorkqv/digitv/lect4.pdf. Last Accessed - September 2009. [cited at p. 10]

[20] Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: System. ISO/IEC 13818-1:2007. [cited at p. 10]

[21] Michail Vlasenko. Supervision of Video and Audio Content in Digital TV Broadcasts. Royal Institute of Technology, School of Information and Communication Technology, COS/CCS 2007-30, December 2007. http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/071225-Michail_Vlasenko-with-cover.pdf. [cited at p. 14]

[22] MPEG-4 Part 14: MP4 file format. ISO/IEC 14496-14:2003. [cited at p. 15]

[23] Smooth Streaming and Fragmented MP4. http://blog.cmstream.net/2009/07/fragmented-mp4-and-smooth-streaming.html. Last Accessed - September 2009. [cited at p. 15]

[24] Fragmented MP4 Structure. ISO/IEC 14496-12:2008. [cited at p. 15]

[25] Content Delivery Networks. http://en.wikipedia.org/wiki/Content_delivery_network. Last Accessed - November 2009. [cited at p. 16]

[26] Amazon CloudFront. http://aws.amazon.com/cloudfront/. Last Accessed - September 2009. [cited at p. 16, 19]

[27] Amazon Simple Storage Service - S3. http://aws.amazon.com/s3/. Last Accessed - November 2009. [cited at p. 16]

[28] Akamai HD Network. http://www.akamai.com/hdnetwork. Last Accessed - December 2009. [cited at p. 16]

[29] Peter T. Barrett, David L. De Heer, and Edward A. Ludvig. Advertisement Insertion, 2009. United States Patent 20090199236. [cited at p. 16]

[30] Digital Program Insertion Cueing Message for Cable. American National Standard - ANSI/SCTE 35 2007, 2007. http://www.scte.org/documents/pdf/standards/. [cited at p. 17]

[31] Martin A. Schulman. Methods and Apparatus For Digital Advertisement Insertion in Video Programming, 1997. United States Patent 5600366. [cited at p. 17]

[32] Reem Safadi. Apparatus and Method for Digital Advertisement Insertion in a Bitstream, 2002. United States Patent 6487721. [cited at p. 17]

[33] Michael G. Perkins and William L. Helms. Non-seamless Splicing of Audio-Video Transport Streams, 1999. United States Patent 5859660. [cited at p. 17]

[34] Peter T. Barrett. Local Advertisement Insertion Detection, 2009. United States Patent 20090320063. [cited at p. 17]

[35] Jen-Hao, Yeh Jun-Cheng Chen, Jin-Hau Kuo, and Ja-Ling Wu. TV Commercial Detection in News Program Videos. *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 5:4594 – 4597, May 2005. [cited at p. 18]

[36] Peter T. Barrett. Advertisement Signature Tracking, 2009. United States Patent 20090320060. [cited at p. 18]

[37] Apple, HTTP Live Streaming Overview. Last Visited -September 2009. http://developer.apple.com/iphone/library/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/StreamingMediaGuide.pdf. [cited at p. 21, 22, 79]

[38] R. Pantos (Editor). HTTP Live Streaming. IETF, Informational Internet-Draft, June 2009. Available from: http://tools.ietf.org/html/draft-pantos-http-live-streaming-01. [cited at p. 22]

[39] Windows Media HTTP Streaming Protocol Specification (MS-WMSP). version 3.1, September 2009. Available from http://msdn.microsoft.com/en-us/library/cc251059(PROT.10).aspx. [cited at p. 23]

[40] Advanced Systems Format (ASF) Specification. Revision 01.20.03, December 2004. http://www.microsoft.com/windows/windowsmedia/format/asfspec.aspx. [cited at p. 23]

[41] Digital Program Insertion. Last Visited - January 2010. http://en.wikipedia.org/wiki/Digital_Program_Insertion. [cited at p. 25]

[42] Digital Program Insertion Splicing API. Amerian national standard - ANSI/SCTE 30 2009, 2009. http://www.scte.org/documents/pdf/standards/. [cited at p. 25]

[43] Cisco Advanced Advertising. Last Visited - January 2010. http://www.cisco.com/en/US/netsol/ns800/networking_solutions_solution.html. [cited at p. 26]

[44] Thales Digital Program Insertion Monitor. Available From http://catalogs.infocommiq.com/AVCat/images/documents/pdfs/DPI_monitor.pdf. [cited at p. 26]

[45] Thomson Digital Program Insertion Monitor. Available From http://www.grassvalley.com/assets/media/2039/CDT-3052D.pdf. [cited at p. 26]

[46] Transport Stream Detector. Available From http://www.norpak.ca/pages/TSD-100.php?L1=TSD-100. [cited at p. 26]

[47] Alcatel-Lucent - Emerging IPTV Advertising Opportunities. Available From http://next-generation-communications.tmcnet.com/topics/innovative-services-advanced-business-models/articles/42283-emerging-iptv-advertising-opportunities.htm. [cited at p. 26]

[48] AD Marker Insertion System. Available From http://www.packetvision.com/dmdocuments/AMIS_Product_Brief.pdf. [cited at p. 26]

[49] Innovid Video Mapping for the ad space. Available From http://www.innovid.com/technology.php. [cited at p. 26]

[50] FFMPEG - A Command Line Tool for Media Conversion. Available From http://ffmpeg.org/. [cited at p. 27]

[51] URL - Uniform Request Locator. IETF Network Working Group- Request for Comments- 1738, December 1994. Available at Availableathttp://www.ietf.org/rfc/rfc1738.txt. [cited at p. 27, 32]

[52] MP4Split- Tool for generating fragmented MP4. Available From http://smoothstreaming.code-shop.com/trac/wiki/Smooth-Streaming-Encoding. [cited at p. 29, 55]

[53] MP4Explorer- Tool for analyzing MP4 file. Available From http://mp4explorer.codeplex.com/. [cited at p. 29]

[54] About Proxy Server. Last Accessed - December 2009. http://en.wikipedia.org/wiki/Proxy_server. [cited at p. 31]

[55] Apache HTTP Client API. Available From http://hc.apache.org/httpclient-3.x/. [cited at p. 32]

[56] MPEG-TS analyzer. Available From http://www.pjdaniel.org.uk/mpeg/index.php. [cited at p. 39]

[57] VLC Media Player. Available From http://www.videolan.org/vlc/. [cited at p. 77]

[58] Mplayer Movie Player. Available From http://www.mplayerhq.hu/design7/dload.html. [cited at p. 77]

[59] Safari Browser for iPhone. Available From http://www.apple.com/iphone/iphone-3g/safari.html. [cited at p. 79]

Appendices

Appendix A

PAT and PMT Table

A.1 PAT and PMT header

A.1.1 Program Association Table

Program Association Table(PAT) contains informaion about the PID values of the transport stream packets which holds the program definition. It also contains the program number to show whether it is network PID or program map PID. PAT consists of the following informations:

- **Table id** is a 8 bit field and set to 0x00.
- **Section syntax indicator** is a 1 bit field which should be set to 1.
- **Section length** is a 12 bit field. First 2 bits are always set to '00' and remaining 10 bits indicate the number of bytes of section following the section lengh.
- **Transport stream id** is a 16 bit field used to identify this transport stream from any other multiplex within the network.
- **Version number** is 5 bit field to indicate the version of PAT table and should be changed along with changes of PAT.
- **Current next indicator** is a 1 bit field to indicate whether the current sent PAT table is applicable or not.
- **Section number** provides the number of this section. The field lenght is 8 bit.
- **Last section number** is a bit field and indicates the number of the last section.

- **N Loop** contains the following information
  - **Program number** is a 16 bit field and indicates the program to which program map PID is applicable. If program number value is 0x0000 then network PID exists, otherwise program map PID exists.
  - **Network PID** is 13 bit field. It specifies the PID values of the TS packets which represents network information table.
  - **Program map PID** is also and 13 bit field and specifies the PID values of the TS packets that contains the program map section
- **CRC32** is a 32 bit that contains the CRC value.

A.1.2 Program Map Table

Program Map Table(PMT) provides the complete collection of all program definitions of Tranpost packet. It contains the following header information.

- **Table id** is a 8 bit field and should be set to 0x02 in the case of a TS program map section.
- **Section syntax indicator** is a 1 bit field which should be set to 1.
- **Section length** is a 12 bit field. First 2 bits are always set to '00' and remaining 10 bits indicate the number of bytes of section following the section lengh.
- **Program number** is a 16 bit field and indicates the program to which program map PID is applicable.
- **Version number** is 5 bit field to indicate the version of PMT and should be changed along with changes of PMT.
- **Current next indicator**is a 1 bit field to indicate whether the current sent PMT is applicable or not.
- **Section number** is a 8 bit field and it should be set to 0x00.
- **Last section number** is also a 8 bit field and it should be set to 0x00.
- **PCR PID** is a 13 bits field and indicates the PID values which will contain PCR in the TS packet.
- **Program info length**is a 12 bit field. First 2 bits are always set to '00' and remaining 10 bits indicate the number of bytes of the descriptor.
- **N Loop** contains the following information in the loop that will iterate N times.

  - **Stream type** is a 8 bit field indicates the type of program elements carried in TS packet.
  - **Elementary PID** is a 13 bit field and provides the PID values that carries the elemenatary stream.
  - **ES info length** is a 12 bit field. First 2 bits are always set to '00' and remaining 10 bits indicate the number of bytes of descriptor.

- **CRC32** is a 32 bit that contains the CRC value.

Appendix B

Fragmented MP4 file for Streaming

B.1 Moving Header Information

Microsoft introduced "Smooth Streaming" that provides seamless bit rate switching of video by dynamically detecting the network conditions. In order to deliver the media, they have used MP4 container. Normally, a MP4 file format contains a header and media data where the header contains the metadata information. In order to use it for streaming the header information should be placed in the beginning. But, the traditional MP4 file has the header information at the last. Figure B.1 below shows the traditional file MP4 file format.

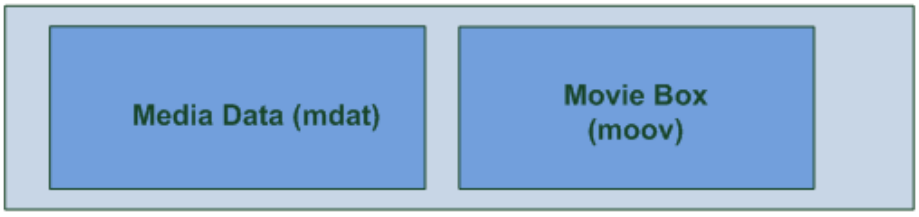


Figure B.1: Traditional MP4 file format

To move the header in the front, we have used qt-faststart that can be found in: /ffmpeg/tools/qt-faststart and listing B.1 shows the command used for altering the header information.

Listing B.1: Moving header information

```
qt-faststart input.mp4 output.mp4
```

The format of output.mp4 will be as figure B.2:

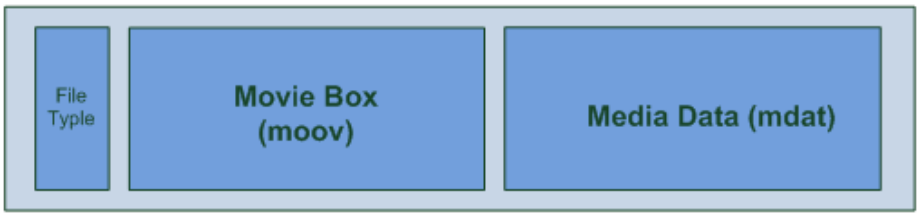


Figure B.2: Traditional MP4 file format

B.2 Transcoding

FFmpeg is used to encode the input video into H264 format where MP4 is used as container. Listing B.2 shows the shell script used for transcoding.

Listing B.2: Shell script - transcoding

```
#!/bin/bash
 outfile="video.mp4"
 options="-vcodec libx264 -b 512k -flags +loop+mv4 -cmp 256 \
-partitions +parti4+parti8+partp4+partp8+partb8 \
-me_method hex -subq 7 -trellis 1 -refs 5 -bf 3 \
-flags2 +bpyramid+wpred+mixed_refs+dct8 -coder 1 -me_range 16 \
-g 250 -keyint_min 25 -sc_threshold 40 -i_qfactor 0.71 -qmin 10\
-qmax 51 -qdiff 4"
  ffmpeg -y -i "imputfile" -an -pass 1 -threads 2 "options" "
      outfile"
```

B.3 Generating Fragmented MP4 and Manifest Files

An open source tool "MP4Split" is used to create server and client manifest files and fragmented mp4 in *.ismv format. Each terminology is represented by "box" in MP4 file format. Each fragment section consists of two parts: Movie Fragment (moof) and media data (mdat). Moof section carries more accurate fragment level metadata and media is contained in the mdat section. Installation of mp4split in ubuntu is done with the commands listed in listing B.3.

Listing B.3: Downloading and installing MP4split

```
wget http://smoothstreaming.code-shop.com/download/mp4split-1.0.2.
    tar.gz
tar -zxvf mp4split-1.0.2.tar.gz
cd ~/mp4split-1.0.2
```

```
./configure -with-expat
make
sudo make install
```

Listing B.4 shows mp4split commands used to generate a client manifest file, server manifest files and fragmented MP4 .

Listing B.4: MP4split commands

```
mp4split - i input.mp4 -o output.ismc
mp4split - i input.mp4 -o output.ism
mp4split - i input.mp4 -o output.ismv
```

There is another open source tool MP4 explorer from cmstream that is used to analyze the header information of a MP4 file. Figure B.3 shows the fragments information of encoded MP4 file. Chunks information are delivered to the client through client manifest file (*.ismc).

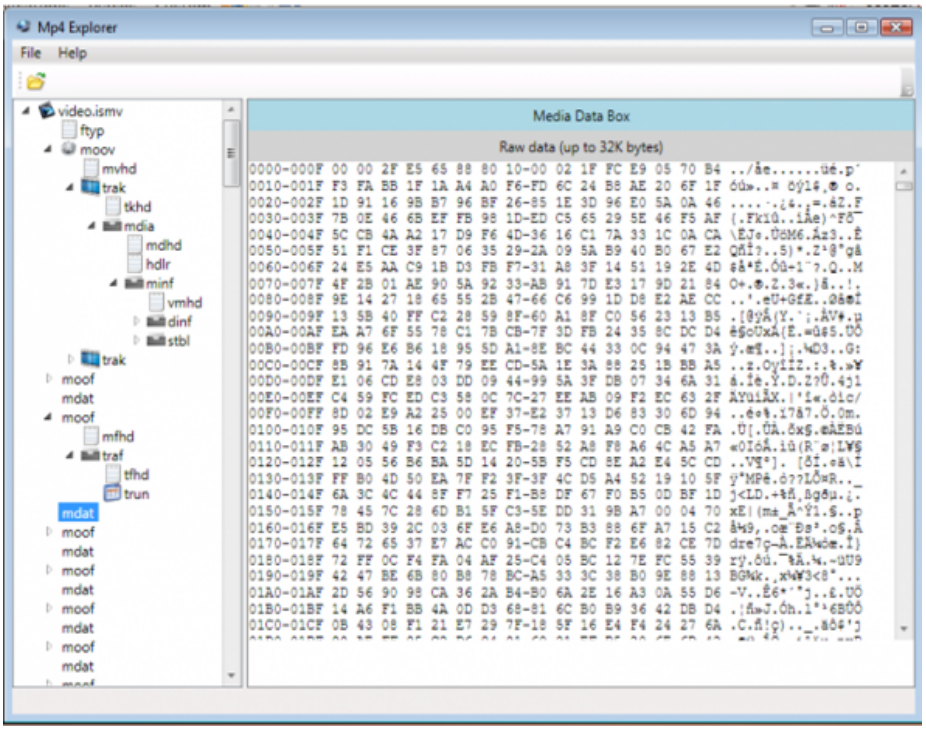


Figure B.3: Fragmented MP4 file in MP4 Explorer

Appendix C

Java Client For Analysis

C.1 Java Client for Concurrent Request

Listing C.1: Java Client

```
package proxyperformanceclient;

import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Date;


class clientThread implements Runnable
    {
        public String[] urls;
        public int id;
        clientThread() {
        }
        clientThread(String[] urls, int id) {
            this.id = id;
            this.urls = urls;
        }
        public void run()
        {

            long time1 = new Date().getTime();
            long time2 = -1;
            HttpURLConnection  uCon = null;
            int responseCode = -1;
            for(String src : this.urls) {
```

```
                InputStream is = null;
                try {
                    URL Url;
                    byte[] buf;
                    int ByteRead,ByteWritten=0;
                    Url= new URL(src);
                    uCon = (HttpURLConnection)Url.openConnection();
                    responseCode = uCon.getResponseCode();
                    is = uCon.getInputStream();
                    buf = new byte[1024];
                    while ((ByteRead = is.read(buf)) != -1) {
                        if(time2<0)
                            time2 = new Date().getTime();
                    }
                }
                catch (IOException e) {
                    responseCode = -1;
                    // e.printStackTrace();
                }
                finally {
                    try {
                        is.close();
                    }
                    catch (Exception e) {
                        //e.printStackTrace();
                    }
                }
            }
            long time3 = new Date().getTime();
            System.out.println("[" + this.id + "] " + responseCode
                + " " + (time2-time1) + " ms "  + (time3-time1) + "
                ms");
        }
    }
public class Main {

    /**
     * @param args the command line arguments
     */


    public static void main(String[] args) {
        int num_Threads = Integer.parseInt(args[0]);
        Thread[] threads = new Thread[num_Threads];
        String[] urls = new String[args.length-1];
        for(int a = 1; a<args.length;a++) {
            urls[a-1] = args[a];
        }
        for(int i=0; i < num_Threads; i++) {
            threads[i] = new Thread(new clientThread(urls, i));
            //threads[i].run();
```

```
            }
            for(int i=0; i < num_Threads; i++) {
                threads[i].start();
            }
    //          Thread myThread = new Thread(new clientThread(args, 0));
    //          myThread.start();

        }

    }
```

Appendix D

System Testing

The implemented solution has been tested with four clients: a laptop, Sony PlayStation 3, Motorola Kreatel STB, and an Apple iPhone. The following subsections describe the testing scenarios based on a client request to proxy, the proxy's request to the content servers and proxy's response to client.

D.1 Scenario 1: Laptop running Microsoft's Windows Vista as a client

The implemented system has been tested with the players listed in table D.1. This section describes the testing with the VLC player only.

Table D.1: List of players used

Name	Description
VLC Player	VLC player is an open source media player that can receive and playout a HTTP media stream [**?**]. VLC player is used to for playout of the media stream at the laptop client.
Mplayer	Mplayer was also used by the laptop client to playout the received HTTP media stream [**?**].

To initiate a request from a VLC player, we requested the proxy server's URL along with the media asset's ID, shown in figure D.1 below.

Figure D.2 shows that the request has been received by the proxy server and fetched the URL and requesting the media from the respective locations.

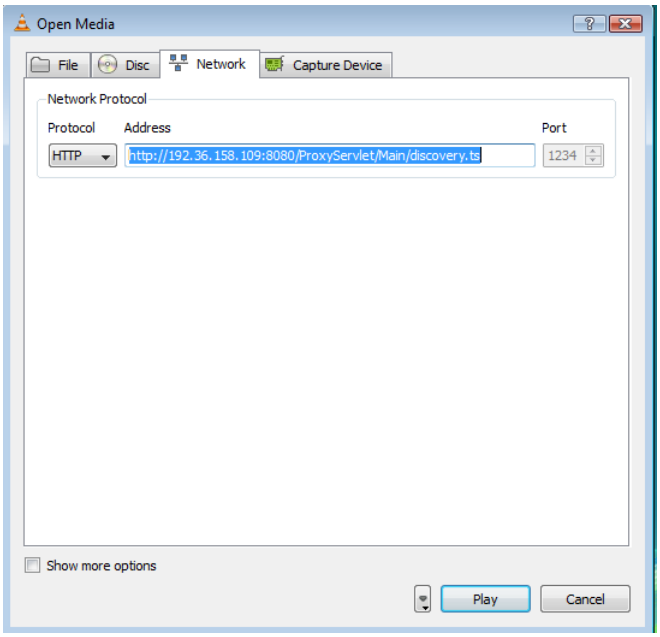


Figure D.1: VLC requesting

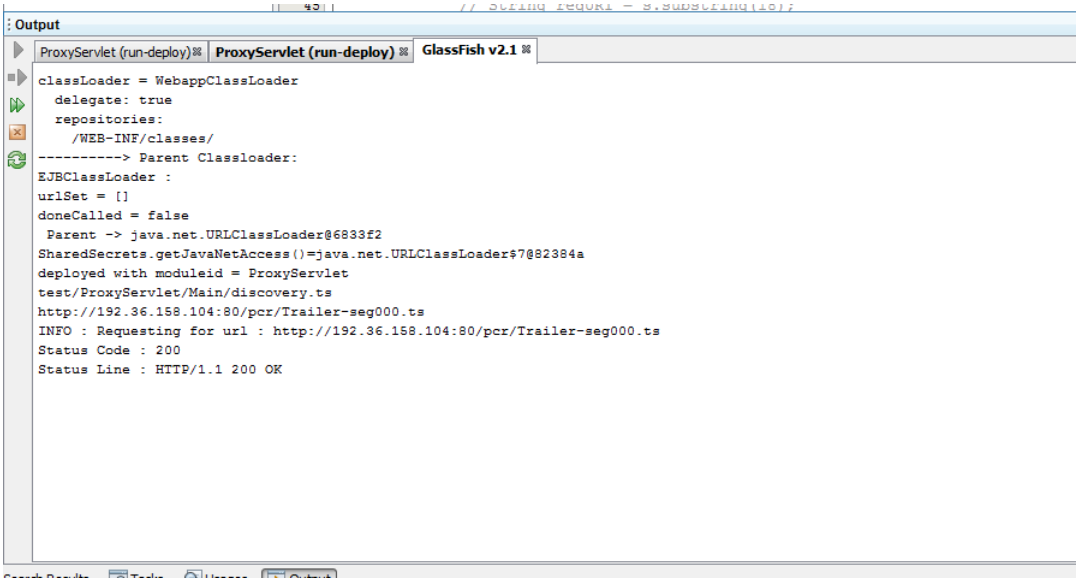


Figure D.2: Proxy Server - URL fetching

D.2 Scenario 2: Apple iPhone as a client

To test the implemented system, we have used an Apple Iphone as a client. The Apple iPhone cannot play MPEG-TS files directly, but it can play TS files through a m3u8 playlist. To test the system, we have placed the URL of the proxy server in a m3u8 playlist, residing in the Ubuntu Server. Listing D.1 describes

the content in the m3u8 playlist. Tag information can be found in Apple IETF draft [37].

Listing D.1: M3U8 playslist format

```
#EXTM3U
#EXT-X-TARGETDURATION:5220
#EXTINF:5220,
http:/proxyserver'sURL/Assetid.ts
#EXT-X-ENDLIST
```

Apple iPhone's Safari browser [**?**] has been used for requesting the URL of m3u8. Figure D.3a and D.3b shows the image of iPhone and request initiation to the proxy server through Safari.

http://severurl/iPhone.m3u8

(a) iPhone 3G (b) iPhone request for m3u8 playlist

Figure D.3: (a) iPhone 3G and (b) iPhone request for m3u8 playlist

Hence, through m3u8 playlist, a request to proxy server is initiated. The proxy streaming server performs the same operation shown in above figure D.2.

D.3 Scenario 3: PlayStation 3 as a client

To initiate the request for the media from the PlayStation. The playstation browser has been used to initiate a request. Figure D.4a and D.4b illustrates the image of playstation and request initiation to the proxy server.

(a) PlayStation 3 (b) Playstation 3 requesting for media

Figure D.4: (a) PlayStation 3 and (b) Playstation 3 requesting for media

The proxy streaming server performs the same operation shown in above figure D.2.

D.4 Scenario 4: Motorola Set Top Box as a client

To test the implemented system with the Motorola STB, first we need to connect to the STB using telnet, then give the command in listing D.2 to initiate the request to the proxy server. Figure D.5 shows the image of Motorola STB.

Figure D.5: Motorola Set Top Box

Listing D.2: Requesting from Motorola STB

```
toish uri LoadUri http://internalserver:8080/stb/video-test.svg?
    video=URLproxy image/svg+xml
```

Appendix E

Test Results

E.1 Test Results

We have calculated the download transaction time for the file size of 10 MB. Initially, we started our test with 1 request, then increased the number of requests in increments of 5 until reaching 46 concurrent requests. Table E.1, E.2, and E.3 summarizes the results with tests of a client requesting one stitched file, client requesting several chunks from the content server through proxy streaming server, and client requesting several chunks from the content server.

Table E.1: Client requesting one stitched file from the content server directly

Concurrent request	Average transaction time (ms)
1	224,6
6	598,8
11	1042,44
16	1517,27
21	1971,44
26	2397,89
31	2867,28
36	3311,56
41	3751,17
46	4182,52

Table E.2: Client requesting several chunks from the content server through proxy streaming server

Concurrent request	**Average transaction time (ms)**
1	314,55
6	883,57
11	1291,57
16	1687,12
21	2092,40
26	2474,98
31	2859,06
36	3243,33
41	3636,73
46	4003,28

Table E.3: Client requesting several chunks from the content server

Concurrent request	**Average transaction time(ms)**
1	237,75
6	762,20
11	1441,16
16	1956,78
21	2460,67
26	2959,65
31	3455,82
36	3961,63
41	4508,33
46	5009,57

We have compared the average time required for a client making request through the proxy when the client requests one file (see figure E.1) and when the client requests multiple files (see figure E.2) from content servers. After analyzing the values, we can see that proxy takes more time to deliver the files when the few concurrent requests in comparison to when there are many concurrent requests.

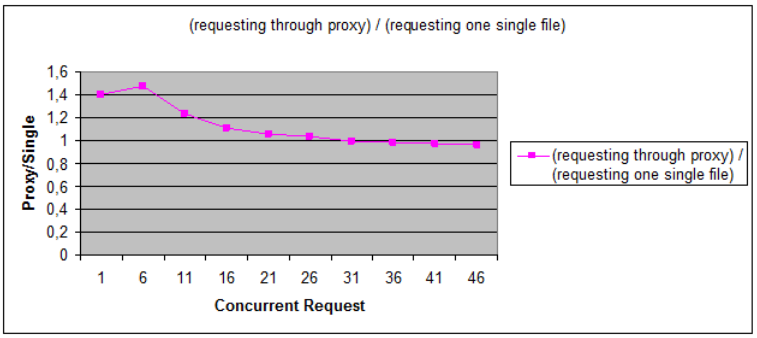


Figure E.1: comparison between request through proxy and request of one single file

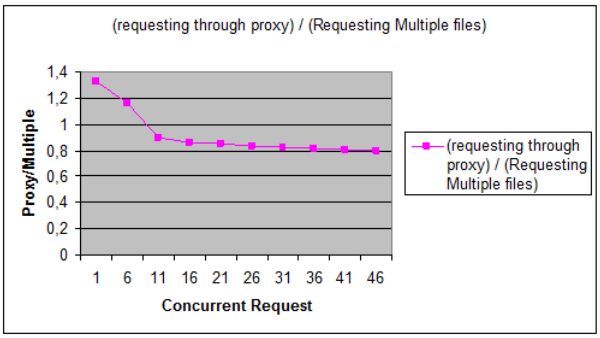


Figure E.2: comparison between request through proxy and request of multiple files

TRITA-ICT-EX-2010:46

www.kth.se