End-to-End Stereoscopic Video Streaming System

by

Selen Pehlivan

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science
in
Electrical and Computer Engineering

Koç University

August 2006

Koc University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Selen Pehlivan

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

M. Reha Civanlar, Ph. D. (Advisor)

A. Murat Tekalp, Ph. D.

Attila Gürsoy, Ph. D.

Date: _____

ABSTRACT

Media streaming has been an active research area where efficient and network friendly media coding techniques, synchronization, bandwidth, packet loss and delay issues, delivery protocols, and, integrated media players and display systems have all been addressed since early 90's. Today, there exist several open source and commercial products that can be used to build an integrated streaming environment supporting various audio and video formats. After the developments of streaming systems that feature classical, 2D video, the attention now is focused on the third dimension; mostly because of the promising progresses in 3D displays and multi-view video coding techniques.

Multi-view video contains views of the same scene from multiple perspectives captured by several cameras. However, only two of these views can be watched by a human viewer at any given time. This brings the idea of developing streaming system delivering stereo video that can be adapted to multi-view by selecting the two views displayed based on the user's current perspective. This thesis presents design and implementation of an end-to-end stereoscopic streaming system. The system has been constructed using the standard protocols: Real Time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP), Session Announcement Protocol (SAP) and Session Description Protocol (SDP) that are also used in well known commercial and open source monoscopic streaming systems. The system architecture is based on independent transmission of two channels of the stereo video in order to achieve selective transmission of mono or stereo video depending on the available bandwidth or the user's receiver and display equipment. The server side designed and developed to implement the basic functionalities required from an RTSP server. On the receiver side, open source VideoLAN Client media player has been extended in order to process and display received stereo video sequences over two logical channels. Also, open source Live555 RTSP library and FFMPEG codec library are integrated into the media

player for extending it to handle stereo streams. The resulting end-to-end platform consists of a pre-encoded stereo-video-streaming server and a media player providing synchronized stereo display on the client side with media delivery over RTP. The system is tested using stereo video sequences compressed using H.264 Extension video codec and a display where end users can view the stereo video using two projectors and polarized glasses.

ÖZETÇE

Duraksız çoğul ortam iletimi aktif araştırma alanlarından biri olmuştur. Bu konudaki araştırmalar doksanlı yılların başlarından beri etkin ve kullanışlı ortam kodlama teknikleri, eşzamanlama, bant genişliği, paket kaybı ve paket gecikme sorunları, teslim protokolleri, gelişmiş ortam oynatıcıları ve görüntüleme sistemleri gibi birçok konuda devam etmektedir. Günümüzde, aktarım sistemlerini oluşturmak için değişik audio ve video çeşitlerini destekleyen birçok açık kaynak kodlu veya ticari ürün mevcuttur. Klasik iki boyutlu video iletimini sağlayan duraksız iletim sistemlerinin geliştirilmesinden sonra ilgi üç boyutlu sistemler üzerinde yoğunlaşmıştır. Bu ilgi üç boyutlu görüntüleme cihazlarındaki ve çok görüşlü video kodlama tekniklerindeki ümit verici gelişmeler sonucunda doğmuştur.

Çok görüşlü video, aynı sahnenin birkaç kamera tarafından farklı perspektiflerden yakalanan görüşlerini içerir. Herhangi bir zamanda bu görüşlerin sadece iki tanesi bir izleyici tarafından izlenebilir. Bu durum stereo videoların duraksız aktarımını sağlayan ve çok görüşlü videolara adapte edilebilen sistemlerin geliştirilmesi fikrini akla getirmiştir. Bu sistemlerin adaptasyonu izleyicinin o andaki perspektifine bağlı olarak ilgili iki görüşün seçilmesi ve gösterilmesiyle sağlanır. Bu tez çalışması dizayn edilen bir uçtan uca duraksız stereo video aktarım sisteminin gerçekleşmesini içermektedir. Sistem gerçek zamanlı duraksız aktarım protokolü(RTSP), gerçek zamanlı ağ protokolü(RTP), oturum duyuru protokolü (SAP) ve oturum tanımlama protokolü (SDP) gibi iyi bilinen ve aynı zamanda ticari ve açık kaynak kodlu mono duraksız aktarım sistemleri tarafından da kullanılan standart protokoller kullanılarak kurulmuştur. Sistem mimarisi mono ve stereo video transferi için mevcut bant genişliği veya kullanıcının alıcı ve görüntüleme kapasitesine göre kendini ayarlayabilecek şekilde düşünülmüştür. Bunu başarabilmek için de sistem iki görüşün farklı kanallar üzerinden bağımsız transferini sağlayacak şekilde kurulmuştur.

Çoğul ortam sunucusu bir RTSP sunucusunun gerektirdiği ana fonksiyonlarla dizayn edilmiş ve geliştirilmiştir. Alıcı tarafta ise iki kanal üzerinden alınan stereo video dizilimlerinin işlenmesi ve gösterilmesi için açık kaynak kodlu VideoLAN Client mono ortam oynatıcısı genişletilmiştir. Bunun yanında, Live555 RTSP kütüphanesi ve FFMPEG kodlayıcı-kodçözücü kütüphanesi stereo bit dizgilerini işleyebilecek hale getirilmiş ve ortam oynatıcısına uygun olarak kaynaştırılmıştır. Sonuç olarak geliştirilen uçtan uca aktarım platformu daha önceden kodlanmış stereo videoları aktaran sunucudan ve RTP üzerinden alınan ortamların istemci tarafında senkronize olarak gösterimini sağlayan bir ortam oynatıcısından oluşturulmuştur. Sistem stereo için genişletilmiş H.264 video kodlayıcı-kodçözücüsü kullanılarak sıkıştırılan stereo video dizileri kullanılarak, izleyicilerin stereo videoları projektörler ve polarize edilmiş gözlükler vasıtasıyla görebildikleri bir görüntüleme sistemiyle test edilmiştir.

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my thesis supervisor Prof. Dr. M. Reha Civanlar. I would like to thanks for his excellent guidance and encouragement throughout my thesis study.

Also, I would like to thank Prof. Dr. A. Murat Tekalp to discuss different aspects of my thesis work. I am grateful for his suggestions during my study.

I would like to thank Prof. Dr. A. Murat Tekalp and Assoc. Prof. Dr. Attila Gürsoy for spending their valuable time to review my thesis.

This work was supported by EC within FP6 under Grant 511568 with the acronym 3DTV.

Finally, I would like to give my special thanks to my family; my mother and my father for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

3DTV	Three Dimensional Television
ISO	International Standards Organization
ITU	International Telecommunications Union
MPEG	Moving Picture Experts Group
AVC	Advanced Video Coding
MVC	Multi-view Video Coding
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
RTP	Real-time Transport protocol
RTSP	Real Time Streaming Protocol
SAP	Session Announcement Protocol
SDP	Session Description Protocol
RFC	Request for Comments
RFC 2326	Real Time Streaming Protocol (RTSP)
RFC 2327	Session Description Protocol (SDP)
RFC 2974	Session Announcement Protocol (SAP)
RFC 3550	A Transport Protocol for Real-Time Applications (RTP)
RFC 3551	RTP Profile for Audio and Video Conferences with Minimal Control
NALU	Network Abstraction Layer Unit
NAL	Network Abstraction Layer
STAP-A	Single-time Aggregation Packet without Decoding Order Number
FU-A	Fragmentation Unit without Decoding Order Number
ATTEST	Advanced Three-Dimensional Television System Technologies
MERL	Mitsubishi Electric Research Laboratories

MPEG-TS	MPEG Transport Stream
FIFO	First In First Out
DSS	Darwin Streaming Server
QSS	QuickTime Streaming Server
VLC	VideoLAN Client
VLS	VideoLAN Server
GUI	Graphical User Interface

Chapter 1

INTRODUCTION

1.1 Problem Definition and Contributions

Multi-view video is an extension of the traditional 2D video, in that it contains multiple perspectives of the same scene at any one instance in time. An ideal multi-view system allows any user to watch a true 3D stereoscopic sequence from any perspective the viewer chooses [8]. In this study, we focused on stereo streaming which can be extended to a multi-view streaming system in the future. The system is focused on the streaming of a stereo video captured from a fixed perspective.

An end-to-end 3D video system should accommodate selective transmission of mono or stereo video depending on the available bandwidth or the user's receiver equipment. A system based on independent transmission of the two channels of stereo video can be used to achieve this purpose. Moreover, such a system can be built by modifying existing platforms developed for regular monoscopic video streaming. We have designed a platform consisting of a pre-encoded stereo video streaming server, and a media player providing synchronized display on the client side. At the display stage, end users view the stereo video by using polarized glasses. Stereo videos are compressed in an efficient way by H.264 Multi-view Extension Codec (MMRG) [19] which is developed based on multi-view video coding (MVC) techniques [7] and streamed using standard real-time protocols. Receivers with proper display equipment and enough bandwidth can view the content of the video built from multiple channels as stereo.

The media server announces the available media files over a multicast address by Session Announcement Protocol (SAP). This protocol is among the existing and well-known service discovery mechanisms. The corresponding information such as codec type, payload type, clock frequency, Real-time Streaming Protocol Uniform Resource Locator (RTSP URL) of the media file and stereo video attribute are all specified using session descriptions based on Session Description Protocol (SDP) syntax and carried inside the SAP packet payload.

The media server is designed and developed as an RTSP Server. However, only the minimum requirements of an RTSP server have been implemented. The coming RTSP requests from the clients are queued and processed by a server component. Server sets up and plays the requested media files for the corresponding client. The RTSP server generates a different RTSP Session for each connected client. A state machine holds the states for each RTSP Session, and processes the transitions upon requests. Server's media transmission unit is designed based on H.264 [24] stereo coded video sequence streaming. If the corresponding file handler is written for other media types, the server can also stream other media file types. However currently, only H.264 media file handler is implemented.

For each RTPS Session there is also an RTP Session which streams the video data over the network. The requested media file is sent over RTP/UDP. H.264 RTP payload format, RFC 3984, is used for packetization. Single Time Network Abstract Layer Unit and Fragmentation Unit w/o Decoding Order Number (FU-A) packetizations defined in the RTP payload format are implemented. Same RTP timestamp values are used to provide synchronization of the corresponding frames of the right and left views.

On the client side, the VideoLAN Client[26], which is a highly portable multimedia player for various audio and video formats such as MPEG-1, MPEG-2, MPEG-4, DivX, mp3, ogg as well as DVDs, VCDs, and various streaming protocols, is modified for H.264 stereo video processing. The media player is capable of retrieving streamed stereo video

sequences from two separate channels, each dedicated to a view. Media descriptions are retrieved by listening the SAP announcements. Session setup and initialization are done by the RTSP protocol. The live555 RTSP library [27] is used for implementing RTSP functionality such as session setup and play request from the client side. It has been extended for reception of stereo views over two different channels. The media sessions of the RTSP library are modified to support stereo sessions. H.264 decoder in FFMPEG library [25], is also modified for compatibility with the MMRG video encoder in order to support the decoding of the stereo videos. All of these libraries are integrated in the media player to provide a complete system compatible with our media server.

The RTP timestamps are used in order to play the received data as a continuous media file in the client side. The decoding and presentation timestamps are derived from the RTP timestamps. The RTP timestamp increment rate between consecutive frames within a view is used to arrange the presentation time of the frames, which is evaluated with respect to the local wall clock at the receiver side. We achieved synchronized video frames set to play 25fps.

In summary, the contributions of this thesis are as follows:

- A media server with basic RTSP functionality is implemented. Server can supply multiple clients with stereo video. It generates an RTSP Session for each client to hold the state of the connection and the media stream.
- Media Server can stream stereo videos which are pre-encoded by MMRG video codec based on H.264 standard [19, 34].
- The system can stream video as 25 fps over RTP. The H.264 media files are packetized and depacketized based on the RTP Payload format for H.264, RFC 3984.
- Stereo video synchronization is achieved by using the RTP timestamp mechanisms. Each related left and right frames of each view are assigned the

same RTP timestamp. On the receiver side, these video frames are displayed at the same time, when the local wallclock of the client side is equal to the estimated presentation time of the frames.

- VideoLAN Client media player, an open source media player application, is extended for stereo video processing. The video decoding order and presentation timestamps are derived using RTP timestamps. Assigned timestamps also provide the relative order of the received frames between different views and within a view.
- A stereo video description is added to SDP by defining a new attribute.
- Live555 library, which can be integrated as a plug-in for VLC, is used for implementation of the RTSP functionality on the client side. It has been extended for our stereoscopic system. Media session within a library opens another connection as a right channel whenever stereo attribute is defined in the session description.
- H.264 decoder inside FFMPEG library is extended to a stereo decoding mode compatible for the decoding of our streamed video files. The corresponding decoder is integrated to the media player. In the system, the decoder is fed by two input buffers, each of which carries video data for a different view. Moreover it has two video output units, each for a different view.
- Simultaneous playing and decoding at the media player is achieved using the multithreaded architecture of the VLC.
- SAP is used as a service discovery mechanism in the system.
- The video on demand functionalities are implemented by using RTSP protocol. Client can request a media file from the server and the server starts to stream the requested file to the specified port of the client upon request.

- An end-to-end stereoscopic streaming system, which can be extended to the multi-view streaming system in the future, is constructed by the integration of the developed media server and media player.
- The media player performance is satisfactory, and comparable with a local playback.
- *Xinerama* feature of the Linux operating system, which extends the existing desktop to twice, is used to overlap the stereo video frames.
- A 3D video-on-demand application is achieved by the system.

1.2 Background

1.2.1 Overview of the 2D Video Streaming Applications

There exists several video communications and streaming applications, which are implemented for various purposes. Applications can be point-to-point, multicast or broadcast based. They can be an interactive or non-interactive. Video streamed using these applications can be encoded in real-time or can be pre-encoded. Such choices influence the design of a streaming application.

There are three criteria which characterize a multimedia application: the number of media involved the types of the supported media and the degree of integration.

Only the first criterion can classify a document processing application supporting text or graphics as a multimedia system [1]. The second criterion divides applications into two groups: time-dependent applications and time-independent applications. While time-dependent applications use sequential presentation-time units for consecutive media objects, time-independent applications use a single time unit for the whole media object [1]. Some authors define a multimedia system as a system that supports the processing of more than one media with at least one time-dependent medium [1]. The third criterion

points to the media integration. In this case, media integration means that different kinds of media can be independent, but played together. The combination of these three criteria defines a multimedia system.

Common form of communication is point-to-point or one-to-one communication. Unicast video streaming over the Internet is an example for this kind of applications. Another form of communication is broadcast which is also known as one-to-all. It delivers the popular content to all receivers at the same time. The third communication form known as multicast, lies between broadcast and point-to-point applications. It is defined as the distribution of content to more than one host but not one-to-all as in broadcast. In the IP multicast architecture, the routers constructs a delivery tree. The packets are transferred over channels only one times and they are replicated by the routers at the branches of the delivery tree. Therefore, multicast applications provide efficient bandwidth usage to perform data distribution. However, deployment of IP multicast is limited and it is not widely available over the Internet. This situation forces researchers to focus on another layer as an alternative to network layer for implementing multicast functionality. Recently, researchers propose the application layer multicast as an alternative to IP multicast. Currently, Narada [2] and Nice [3] protocols are among the most promising application layer multicast protocols.

An important part of any multimedia system is media encoding. For example, video may be captured and encoded in real-time at the sender side, or it can be pre-encoded and stored for later streaming. Interactive applications such as video conferencing are examples for real-time encoding. In real-time encoding, sender may read uncompressed media data into a buffer. Media data may be audio samples or video frames captured by a multimedia device. Buffer is used by the encoder to produce compressed frames. Data may be compressed in several ways depending on the compression algorithm used as it will be

discussed later. After compression, sender generates network packets from the compressed data for transmission.

Some applications use real-time encoders and decoders, including the adaptive use of error-resilience tools. However, the use of real-time encoders limits the maximum computational complexity for the remaining parts of the system. When streaming from pre-recorded compressed media data, media frames are passed to the packetization routine directly. The remaining processes are the same with the real-time streaming. The main disadvantage of the pre-encoded media file is its limited flexibility. Pre-encoded data can not be significantly adapted to the channel conditions such as different bit rates or to the client display support [4].

Applications are also classified as interactive and non-interactive applications. For interactive applications, a low end-to-end delay is required. The end-to-end delay is the time taken for capturing, encoding, transmission, decoding and display. The end-to-end delay for non-interactive applications is much looser. Thus, interactivity condition of an application has an effect on its design.

1.2.2 Video Compression Algorithms

Raw video must be compressed before transmission. Since raw video uses a large amount of bandwidth, compression is employed for an efficient transmission. The data compression is based on the redundancies and similarities in a typical video signal. There are two kinds of redundancy. Within a single frame there exists spatial redundancy. The nearby pixels of a frame are correlated. In addition to that, consecutive frames of a video sequence hold temporal redundancy, since they contain the same object with small location differences depending on the motion.

To handle spatial redundancy, an image is split into small blocks and decorellating transforms, such as the cosine transform is used. The most effective method to utilize the similarities between the consecutive frames is to predict a frame from the previous frames and then to code the errors in this prediction. The idea is based on the objects, which are same in successive frames. In some conditions, the only difference is the location of the objects. Therefore, estimation of the motion between frames improves the prediction. The process of estimating the motion between frames is known as motion estimation.

With respect to frame prediction, the coded frames are classified into three groups: I frames, B frames and P frames. I-frames are coded independently and they are not dependent on any other frame within a video sequence. B-frames are also called bi-directional frames. They use both previous and future frames in order to be predicted. P-frames only depend on the previous frames.

Video compression standards are developed under the support of International Telecommunications Union-Telecommunications (ITU-T) and the International Organization for Standardization (ISO). The first video compression standard to gain widespread acceptance was the ITU H.261 adopted as a standard in 1990. In 1993, the ITU-T initiated a work item with the primary goal of video telephony over the public switched telephone network (PSTN). The video compression portion of the standard is H.263. An enhanced H.263 was finalized in 1997.

The Moving Pictures Expert Group (MPEG) was established by the ISO in 1988 to develop a standard, called MPEG-1, for compressing video and associated audio. A second stage of their work is known as MPEG-2 which was an extension of MPEG-1. It was developed for application toward digital television and for higher bit rates [14]. A third standard was called MPEG-3, but later on it was wrapped and addressed by MPEG-2. Lastly, MPEG-4 was constituted as the third phase of their work. It provided improved

compression efficiency and error resilience features, as well as object-based processing, integration of both natural and synthetic content, content-based interactivity [4].

The H.264 standard is finalized by the Joint Video Team (JVT), from both ITU and ISO MPEG. It achieves a significant improvement in compression over all prior video coding standards, and it is also called MPEG-4 Part 10, or Advanced Video Coding (AVC) [4].

1.2.3 Media Synchronization

As we stated before, one criterion which distinguishes multimedia applications from other applications is media integration. Various media streams should be rendered together in a synchronized manner. Media synchronization refers to sustaining temporal relationships between various media streams and within one media steam. Temporal relation is defined by the temporal dependencies between two or more media objects that are recorded at the same time. This time based relation is what we mean by synchronization [1].

Media streams are also defined as time-dependent media objects in which presentation durations of each unit are equal. Time-dependent media objects are distinguished with respect to time relations. Intra media synchronization is defined for the continuity of presentation units within a media object. Inter media synchronization is used for synchronizations of different streams of media. Without using inter media synchronizations, the skew between streams can be intolerable. A media play-back can be accepted as synchronized if synchronization errors are on the order of 80 ms to 100 ms [1] which are below the limit of human perception.

Media objects exhibit a more complicated synchronization problem when streamed over a network instead of played locally. There are three solutions for the transmission of media synchronization information. One of them is to send the synchronization information to the

client side prior to the actual media transfer. But this has a disadvantage due to the delay of synchronization information transmitted. Another approach is to use an additional channel for synchronization. Its disadvantages are the possible loss that may occur in the additional channel and the allocation of resources for this channel. Multiplexing media streams is another way to solve the synchronization problem. Corresponding audio sequence and video frame are packetized together before transmission. MPEG defined a bit stream which uses such kind of media synchronization by the way of combining video, audio and corresponding synchronization information. This bit stream is viewed as one media stream.

A widely used specification for temporal relations of a time-dependent media is axes-based specification which is also known as time-stamping. At the sender side, a stream is time-stamped to keep temporal information within stream and with other streams of a media. At a sender, compressed video segments are assigned a timestamp and loaded into RTP packets ready for transmission. At the receiver, the received streams are presented along with their temporal relations specified by time-stamps [5].

1.2.4 Playout Buffer

Multimedia applications need a structure to smooth the timing variations caused by the network jitter. For that purpose, playout buffers are used on the receiver side of the applications. Frames are held in the playout buffer for a period of time to solve the smoothing problem. Playout buffer also allows receiving the fragmented packets from network, and grouping them before sending to decoder.

Media clients have commonly a 5 to 15 second buffering before playback. Buffering provides important advantages. One of them is jitter reduction. End-to-end delay for a packet transmission fluctuates from packet to packet. This variation caused by queuing delay or link-level retransmission must be solved to play the media at a constant rate on the

client side. This problem is solved by buffering the received packets for a small period of time before the playback starts. Another advantage of playout buffer is error recovery. A 5 to 15 seconds buffering extends the presentation time for the media data. This extension gives the system a time to retransmit the lost packets. This is important for UDP based applications in which retransmission is not handled by an underlying network protocol such as TCP. Error resilience is another advantage of the buffering. Extension in the presentation deadlines can be used to isolate the losses. Especially, audio data losses can be better concealed with respect to video frames. [4]

1.2.5 Media Transmission

Video delivery can be established by one of two ways. One of them is the file download method and the other one is video delivery via streaming. Downloading is probably the most straightforward method for video delivery. The difference of a video download with a classical file download is the download duration. Since video files are large files, they require more time to finish the download process. This method also brings the storage problem. The client side must have a large storage space to download such a large file before playback. The next approach is streaming video. In video streaming, the compressed video is partitioned on the sender side to transmit. Then on the receiver side, received video packets are decoded and played while the video is still being delivered. This means that video streaming enables simultaneous delivery and playback of the video. Streaming approach overcomes all disadvantages of the download method with low storage space and low delay requirements.

Transport protocol family includes UDP, TCP, RTP and RTCP protocols for media streaming. UDP and TCP provide basic transport functionalities. RTP and RTCP runs over them. TCP protocol maintains congestion control, error control, flow control and

multiplexing functionalities. Although TCP supports reliable transmission, UDP is preferred for streaming media applications. Re-transmission property of TCP causes delays that are not acceptable by streaming applications. Therefore UDP is employed as a transport protocol for video streaming.

RTP is an Internet standard protocol for real-time applications [20]. It supports time-stamping to synchronize different media streams, sequence numbering for correct ordering, payload type identification for classifying the content of media packets and source identification by Synchronization Source Identifier (SSRC) to distinguish different media source. RTCP provides QoS feedback mechanism using sender and receiver reports used for quality of reception reporting. Sender can adjust its transmission rate depending on these reports.

MPEG-2 Transport Stream

MPEG-2 provides a new packet multiplexing format which uses 188 byte as a fix-length packet. This multiplexing format provides the carriage of multiple audio, video and private data streams together. MPEG-2 specifies a more suitable transmission format for error prone environments which is also a more advanced than Program Stream scheme defined in MPEG-1. This new scheme is called Transport Stream. Both these streams use Packetized Elementary Stream (PES) as a common building block formed by packetizing the continuous data produces by encoder.

Transport Stream forms 188 bytes packet which consists of a 4 bytes header together with 184 bytes payload. This makes it more suitable for hardware processing and is powerful for error correction method.

Multiplex mechanism of Transport stream provides correct timing for decoding and display of the video and audio at the receiver side. Audio and video sample clocks are

derived from the Program Clock. This allows the synchronization of corresponding audio and video samples, also called the lip sync mechanism. Mechanism uses time-stamp in the bit stream. Three kinds of time-stamps are used in the Transport Streams: the Program Clock Reference (PCR), Decoding Timestamp (DTS) and Presentation Timestamp (PTS). Program Clock is the equivalent of System Clock Reference used in the Program Stream structure of MPEG-1 for the same functionality. PCR values are used to produce the same clock frequency as the System Time Clock at the decoder. DTS tells the decoder when the data must be removed from the decoder buffer and PTS tells the decoder when the data must be displayed [38]. In our project, we want to stream two streams separately without multiplexing them. Therefore we do not prefer to use this file format. Its multiplexing format brings a heavy load over sides. Moreover 188 byte packets are too small for our case.

MPEG-4 File Format

MPEG-4 standard consists of several parts. Part 14 is the MPEG-4 File format standard which is the delegated multimedia container format for MPEG-4 content. It is based on Apple's QuickTime container format. In 1998, the ISO approved the QuickTime file format as the origin of the MPEG-4 Part 14 container.

The basic data units in QuickTime file is the atom structures. A QuickTime file consists a collection of atoms. QuickTime file format can be streamed using real-time protocols, such as RTP. This requires a streaming server which uses designated information about how to packetize each track in a movie. This designated information is called hinted tracks. These additional tracks are used by the streaming servers for the formation of packets. For different transport types, several hint tracks can be appended to the file. Existing media can

be easily made streamable by adding a hint tracks for the corresponding transport layer. Hint-tracks are only used when streaming a file over a real-time streaming protocol [30].

1.2.6 Mono Streaming Applications

Mono streaming applications became among popular commercial products nowadays. Darwin Streaming Server is one of them. It is Apple's open source streaming server and proxy. It supports RTSP/RTP serving of files in QuickTime File format. Its client application is QuickTime player. The system uses RTSP as control protocol and RTP for real time streaming of media files [30].

Another commercial product is Helix server. Helix server runs on a wide range of operating systems, including Windows and many UNIX variants such as Linux and Solaris. Helix Server is the only digital media server with universal support for live and on-demand RTSP/RTP delivery of all major file formats, including RealAudio and RealVideo, Windows Media, QuickTime, MP3, 3GPP (H.263 and H.264), AAC and AAC+ [39].

Sun Streaming Server is a server implementation of RTSP, RTP/RTCP, and SDP [40]. It delivers on-demand streams from hinted MPEG-4, hinted Apple QuickTime and hinted 3GPP files. Live streaming is also supported via reflection of RTP sources.

VideoLAN Client is a multimedia player which can also be used as a server to stream in unicast or multicast on a high-bandwidth network [26]. It provides media streaming using MPEG-TS file format over RTP. It also supports RTSP functionality with plugin libraries. Media player part can be used with Darwin streaming server as an end-to-end streaming system for the streaming of H.264 media files over RTP.

1.3 Conclusion

In this chapter, we give outlined the main contributions of this thesis and briefly defined its main components. Also, an overview of the mono streaming systems such as synchronization issues, transmission, playout buffer usage, and compression algorithms along with the available commercial streaming systems is introduced.

The next chapters of are organized as follows: In Chapter 2, the recent research on 3D streaming which is related to our work is described. The related standards which are used in commercial mono streaming systems and also in our platform are described in Chapter 3. The architecture and implementation details for the end-to-end stereoscopic streaming system are given in Chapter 4. Lastly, in Chapter 5, we give the test results and summarize our conclusions together with future plans.

Chapter 2

3D VIDEO STREAMING

2.1 Introduction

Three Dimensional (3D) televisions are expected to be among the latest popular technological developments in the entertainment world. The recent achievements in research triggered an increasing interest in 3D audio-visual technologies. Also, promising achievements in the area of auto stereoscopic 3D display technologies increase the popularity of the subject. These developments have generated new ideas around communication applications including Internet, 3D TV Broadcast, and 3D Cinema etc. Additionally, interest in this area also brings the development in all sub-areas of the subject including image capture, 3D representation, view synthesis, compression, transmission, interactive rendering and 3D display. The goal has just become to construct and integrate the best affordable system which provides the whole processing chain of a 3D system with promising quality. As a result the research is focused on a system which is called as 3D TV.

Firstly, we can differentiate between 3D video and 3D TV in order to understand the difference between the terms. 3D video usually refers to geometrically calibrated and temporally synchronized stored video data, while 3D TV includes real-time acquisition, coding, transmission and rendering of dynamic scenes [2]. Therefore 3D video is only the transmitted data of a 3D TV system. The whole scheme of a 3D TV system can be as in the following figure:

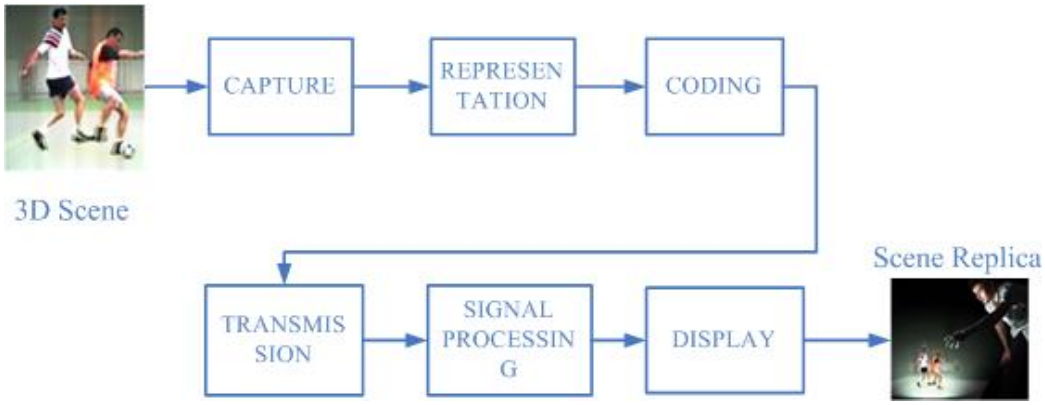


Figure 2.1 Processing Chain of a typical 3DTV system [9]

Figure 2.1 shows the processing chain of a general 3DTV system. 3D scene representation is the most important part among the system units because it effects all other units. 3DAV group, ad hoc group on 3-D audio/video coding established by MPEG committee, includes five central categories for scene representation of a video [6].

- Panoramic Video: This is an extension of a classical planar 2-D image plane to a spherical or cylindrical image plane. Any 2-D view can be reconstructed from such a video sequence.
- Interactive Stereo Video: This includes two views one for left eye and other for right eye. This produces a 3D impression on the viewer.
- Interactive multiple view video: This includes N camera views. The scene is captures by N cameras. To increase the navigation through the scene, it also keeps additional information such as scene geometry.
- 3D video objects: This includes 3D video objects created from available views of a scene captured by multiple cameras. 3D video objects involve shape and appearance.

- 3D audio: This is the audio data part of the 3D sound scene.

After construction of the scene representation using one of the categories, these representations are broadcasted subsequent to efficient coding. This transmission also differentiates the system from a classical 3D video and it is the first step through 3D TV. Increasing number of views brings the problem of high data rate which consumes too much bandwidth. The key point is to code high data rate efficiently in order to send all the views to the receiver nodes using the available bandwidth. Our focus in this chapter is towards the streaming using efficient compression and coding techniques used in multi-viewpoint video and stereoscopic video sequences.

2.2 Multi-view Video Coding Techniques

The efficient video coding techniques play a central role for transmission over a channel with bandwidth limitation. Especially for multi-view videos coding efficiency becomes much more important, proportional to the increasing view. Multi-view video consists of N views, each captured by a camera. Each camera captures a different view of a scene. However, these views contain some amount of redundant data for transmission. Efficient compression can be achieved by removing these redundancies.

As the first step to multi view coding (MVC), investigation of single view image coding techniques is a good starting point. This is because of the fact that multi-view videos contains temporal redundancies caused by time domain changes within the same video sequence and also contains redundancies between corresponding frames of different cameras. Motion compensation in the time domain is about the single view video coding techniques and disparity between cameras corresponds to multi-view image coding.

If we examine the standard video case, the redundancies can be inside a frame or between consecutive frames. To eliminate these redundancies, there exist some techniques used in single view video coding. These techniques may be applied to Multi-view videos with some extensions. The most popular method is the hybrid coding used in single view video encoders such as H.261, H.263, MPEG-1 and MPEG-2 [7]. These encoders uses intra frame and inter frame redundancies to achieve the coding efficiency. Another method used for single view video coding is content based video coding which can also be a candidate for MVC.

2.2.1 MPEG2 Based Multi-View Video Coding

MPEG-2 specifies a coding process for a single view video sequence. It uses the block matching correlation algorithm to eliminate redundancies of a video sequence. In addition to that it offers an applicable solution for two sequence stereoscopic video. However it is not practical for compression of more than two viewpoints. [8] proposes a method which is an extension to the MPEG-2 standard to achieve the compression of multi-view video sequences together with transmission of them over bandwidth limited channels through efficient coding.

This new codec proposes selecting the viewpoint which has the highest correlation of objects with other views. Therefore, the central viewpoint is taken as the main view and other viewpoints are predicted from it [8].

2.2.2 Group-of-GOP (GoGOP) Prediction

In this approach all GOPs are classified as two groups, as Base GOP or as Inter GOP. A picture in the Base GOP can be predicted from reference pictures within the same GOP. However, a picture in an Inter GOP refers to the other pictures both from its own GOP and from other GOPs.

2.2.3 Sequential View Prediction

In the sequential view prediction, each view can refer to the previous view sequence for prediction. In that approach, initial video sequence is predicted from frames within itself. The second video sequence uses its own frames together with the corresponding frame of the first sequence for prediction. Similarly, a frame from third video sequence uses corresponding frame of second video sequence and the frames of its own sequence for prediction [7].

2.2.4 Checkerboard Decomposition

In that prediction method, the even frames of even numbered cameras are set as "low-pass" frames and others are set as high pass frames. Low-pass frames are generated based on lifting structure, whereas high-pass frames are generated using complementary operations. The sequence of low-pass frames for each camera is encoded and sequence of high-pass frames for a given camera is predicted using motion compensation and disparity compensation from the neighboring low-pass frames before encoded [7].

2.2.5 View Interpolation

This approach uses two methods applied to sequences for prediction of multi-view video sequences. The views are divided as even and odd numbered video sequences. The odd numbered video sequences are coded using AVC [34] codec. Then, the even numbered video sequences are predicted by applying one of the view-interpolation methods such as adaptive filtering, table-lookup [7].

2.3 Stereo Video Coding Techniques

Stereo video coding can be thought as a basis for multi-view video coding. It consists of views captured by two cameras. These views are encoded using techniques for stereo video compression. Existing single view video coding techniques may also be used for stereo video coding.

Extension of single view video coding to stereo videos can be categorized into two types [9]. In the first approach, the left and right video sequences are coded independently. In the second approach, the investigation of correlation between left and right frames plays a role in coding. In [10], a method based on single view video compression is introduced. In that method one of the views is coded using MPEG-2 coding. Then other view is estimated both from this coded view by the way of the disparity vectors and from the previously coded frames within the same view. Another approach is proposed in [11]. In this paper, left view is set as base layer and right view is set as enhancement layer. Algorithm only codes base layer, left frames, with MPEG-4 video encoder. Instead of coding right frames, disparity vectors are found and coded. System transmits only the coded left frames and the disparity vectors to decoder side. This provides low bit rates of transmission.

Object base coding is another approach for coding stereo videos as an alternative to block base coding. Here, I will give the structure of an Object-based Stereoscopic codec. These codecs consist of two units: analysis and synthesis units [9]. Analysis unit divides the scene into its objects which are represented by a parameter set. This parameter set corresponding to one object contains object shape, boundary and motion together with depth, texture and color information. Unit controls complicated operations such as image segmentation and motion/structure estimation handled by distinct modules. In segmentation module, the pixels are assigned to their related objects; and in the motion/structure module,

motion and 3D shape is modeled to be used for the estimation of parameters of the object model. The derived parameter sets are encoded. Especially for encoding of object boundary parameter, a large percentage of available bit rate is required. Therefore, parameters must be encoded efficiently. The encoded parameters are then used in the synthesis unit. The reconstructed image is compared with the original one to find approximate errors. Object-based stereoscopic coders contains both analysis and synthesis units in their encoder side, but they contains only the synthesis units in the decoder side.

2.4 Related Work

We can divide this section into two groups as multi-view codec and multi-view systems. In the codec section, the latest multi-view codec will be dıscussed and in system part, 3D video streaming applications will be outlined.

2.4.1 Multi-view Codec

A multi-view codec is introduced in [15,17] as part of the ATTEST project. It uses a view sequence and depth map information. All views are coded using this information. Another multi-view codec is presented in [18]. The approach is extending an available H.264 video codec for multi-view videos. Results show that performance is comparable with the simulcast case in which each video sequence is encoded separately.

In addition to these proposed multi-view video encoders, [19] presents a new multi-view video codec. Compression algorithm is again based on an extension of classical H.264 video compression algorithm. Algorithm changes the buffer structure in such a way that it can code multi-view videos. Corresponding frames of different views are located into the buffer successively according to the distance between camera positions. The idea is that the closest the camera, the higher the estimation accuracy between frames [19]. Then, the resultant video frames queued as mentioned are send to the H.264 standard codec.

Proposed codec presents four reference modes shown in the figure. The modes are also defined related to the closeness of the frames.

The proposed encoder, also called MMRG H.264 Multi-view Extension Codec, in [19] is used for our stereo streaming system. Streamed videos are encoded off-line at mode 1 which supports encoding of stereoscopic video sequences. The NAL Units are encoded such a way that the left frames of the stereo video are encoded within the sequence and right frames of the video are encoded using left NAL Units as reference frames. However, decoder part is too slow to be used for real-time applications. Therefore codec is used only for off-line video encoding. Another H.264 decoder under FFMPEG library is modified in order to decode stereoscopic videos for our application.

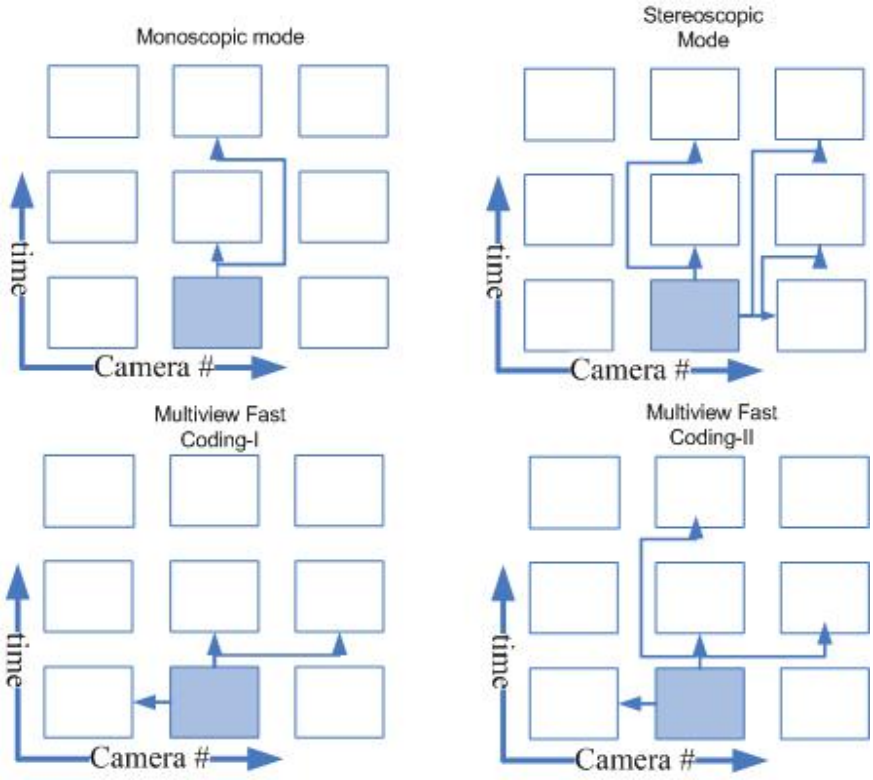


Figure 2.2 H.264 Multi-view Extension Codec, MMRG, modes [19]

2.4.2 Multi-view System

[14] proposes a communication framework for distributed real-time 3D system. System provides 3D video reconstruction, rendering and transmission of each consequent stream according to the analysis in the changing network conditions. 3D video processing is applied to the captured N views of the same scene by calibrated cameras placed at different positions of a scene. In each camera view, the fixed background is deducted. The reconstruction process transforms the 2D pixels into 3D point samples using the geometry information provided by silhouettes.

In this system, the reconstruction process of the previous frames gives a feedback about the performance measures. 3D video frame rate can be adjusted and improved by down sampling the texture information. For transmission, the system uses RTP/RTCP protocols for real-time streaming.

Another approach for 3D TV attempts is ATTEST (Advanced Three-Dimensional Television System Technologies) project. Its main method is based on a 3D data representation format consisting of monoscopic color video and associated per-pixel depth information. Monoscopic color video is coded using MPEG-2 codec and the additional per-pixel depth information is coded by MPEG-4 or H.264/AVC codec. As a result, 3D data are multiplexed together using MPEG transport stream and streamed in real-time. The purpose of using MPEG-2 is to create a system which is compatible with today's 2D digital TV [17]. On the receiver side, stereo or multi-view images are generated using image-based rendering [15]. The authors also analyzed which coding standard is the best one for compression of the depth information. Tests are done using three popular compression standards: MPEG-2, MPEG-4 and H.264/AVC. The results show that H.264/AVC gives the best performance results [15].

Another 3DTV project is MERL. It uses an array of cameras, clusters of network connected computers, and a multi-projector 3-D display system. In this system, multiple

video streams are individually encoded and send over the network. The system acquisition stage includes a set of cameras, which are hardware synchronized. Small camera clusters are connected to the computers. Captured views are compressed using MPEG-2 encoding. Then, compressed video streams are broadcast on separate channels. On the receiver node, each decoder connected to the computer clusters decompress the received bit streams. The decompressed videos are rendered to the display. 3D display is a multi-projector 3D display with lenticular sheets. 16 NEC LT-170 projectors with 1024x768 resolution are used. Each viewer corresponds to a projector in the display [13].

The success of this system is based on broadband network existence. Such a network affords very large bandwidth and almost zero delay to provide immediate decoding on the receiver node. A gigabit Ethernet supports the system requirements [13].

2.5 Conclusion

In this chapter, we have covered the general 3D concepts including proposed compression techniques for multi-view and stereo videos. These techniques are used for 3D streaming in order to increase the streaming capability. Increasing number of views brings the problem of high data rate consuming too much bandwidth. Therefore, efficient video coding in order to send all the views to the receiver nodes over available bandwidth is one of the crucial points for 3D streaming. The chapter has also mentioned about the last related 3D streaming platforms offering a complete system including 3D video capturing, transmission and display.

Chapter 3

RELATED STANDARDS

3.1 RFC 3550: Real Time Transport Protocol (RTP)

This section describes RTP, the real-time transport protocol. RTP is used in real time applications for end-to-end network data transmission such as audio or video over network through multicast or unicast.

RTP Data Transfer packet consists of four parts:

- *Compulsory RTP Header*
- *Optional Header Extension*
- *Optional Payload Header*
- *Payload*

The compulsory RTP header is 12 octet long. The fields in the compulsory header are payload type, sequence number, timestamp and synchronization source identifier. Additionally, there is a field for version number, padding, header extension, marker for significant events. The entire packet is wrapped as a payload for a lower layer protocol. RTP packet structure is as follows:

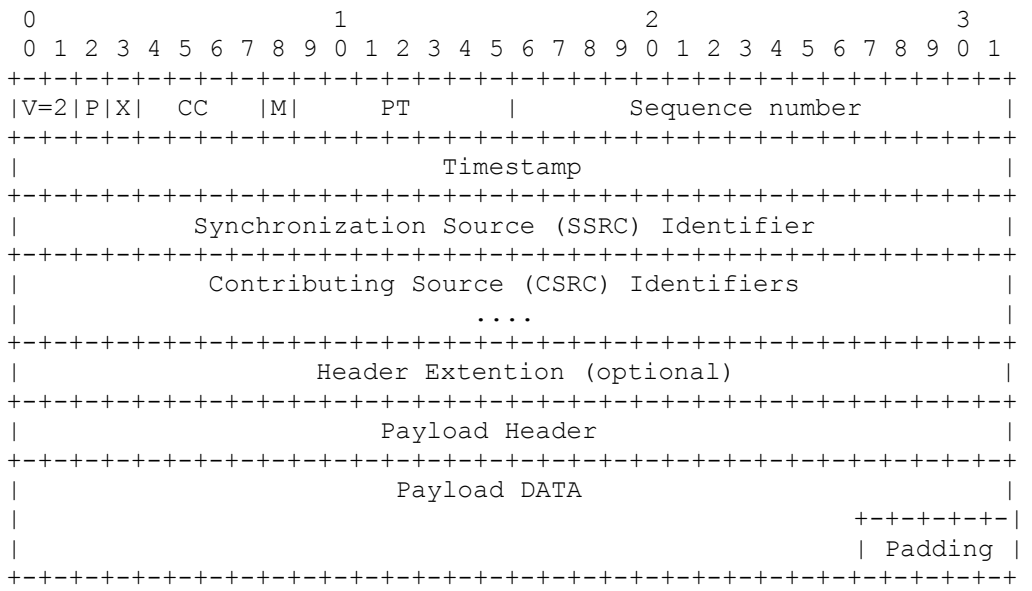


Figure 3.1 RTP Packet Format

The details of the RTP packet format can be found in RFC 3550 [20]. Here, three main fields will be mentioned:

Payload Type Field: Payload type is represented by 7 bits field. It identifies the media transported by RTP. RFC 3351 specifies assignments of payload type codes to payload formats. According to the RFC, there are static and dynamic payload types. The static payload types are pre-defined types assigned to specific media formats. However; due to increasing number of media formats, a range of numbers is reserved for dynamic usage which can be used by any data format provided that the payload type-encoding binding is defined by a mechanism such as Session Description.

Sequence Number Field: 16 bits are reserved from the RTP header for the field. Sequence number is used to arrange the arrived packets which are out of order and also used to detect packet losses over network.

Timestamp Field: Timestamp field indicates the sampling time of the first octet of the RTP data packet. Its value is incremented depending on a specific clock frequency which depends on payload type. Timestamp values may be used for media synchronization. If two media stream has the same encoding, the same clock is used for both of them. However, if their payload types are different such as an audio and a video streams, two different media clocks with different frequencies are required. In the last case, the correct match between samples of media streams for synchronization can be provided by sending RTCP reports to the sides periodically.

SSRC: The SSRC field identifies the synchronization source. It uses 32 bits of the RTP header. This value is selected randomly and no two synchronization source has the same SSRC identifier.

3.2 RFC 2326: Real Time Streaming Protocol (RTSP)

Real Time Streaming Protocol performs as a network remote control. The client side can direct server side via protocol methods. It either controls a single or several streams of continuous media such as audio or video.

Protocol supports the following functionalities defined in [21]:

- *Retrieval of a media file from media server*
- *Invitation of a media server to a conference*
- *Addition of a media to an existing presentation*

While RTSP via TCP is used as a control protocol, the actual data can be transmitted over a different protocol such as RTP/UDP. Required network destination address and port are defined, before data transmission starts. Three different alternatives are proposed for address assignment. Firstly, the transmission can be achieved as unicast. In that case, the media is requested by the client from the server. Client arranges its port numbers and informs server about the ports to be streamed. This alternative can be used for on demand systems. Other two alternatives are suggested for multicast. If the presentation is being multicast, the address can be chosen by client or server. A media server can select the multicast address and port. This can be used for near-on-demand transmission. Otherwise, client side can chose the multicast address.

RTSP States

A media stream may be requested by separate RTSP requests at different times of media server lifetime. This requires the state information for each session. The corresponding session state holds the current condition of the stream for that media session. Moreover, each RTSP request can trigger a state transition within a session. The session changes its states with respect to its current state and the upcoming RTSP request. These requests are:

SETUP: Causes a server to allocate resources for a new RTSP session and the media stream.

PLAY: Starts to send data of allocated media stream to the client specified by the RTSP session properties during SETUP request.

PAUSE: halts the current stream transmission for the corresponding RTSP session.

TEARDOWN: frees the allocated resources for a RTSP session. The streaming is finished.

Other RTSP methods can be found at Table 3.1.

Table 3.1 RTSP methods

Options	• specifies the options.
Describe	• retrieves the description of the presentation
Announce	• posts the description of the presentation whenever send by client • updates the session description in real-time whenever send by server.
Get-parameter	• retrieves the value of a parameter of a presentation or stream specified in URL.
Set-Parameter	• sets the value of a parameter for a presentation or stream specified by the URL.
Redirect	• informs the client about a connect to another sever location.
Record	• initiates recording process of a media stream.

RTSP URL

The network resources using RTSP protocol are specified by “rtsp” or “rtspu”. If the RTSP protocol runs over a reliable protocol such as TCP, “rtsp” is used. Otherwise, “rtspu” is used. The syntax of a RTSP URL is as follows:

rtsp_URL = ("rtsp:" | "rtspu:") "//" host [":" port] [abs_path]

host is the IP address of the side where the file is located. *Port* number may also be specified in the RTSP URL. Otherwise 554 is the default value. Then the *absolute path* of

the media is added to the URL to identify the media located on the server. URLs may refer to a stream or an aggregate of streams such as a presentation.

3.3 RFC 2974: Session Announcement Protocol (SAP)

There are some mechanisms to announce an available media session. One of these mechanisms is Session Announcement Protocol (SAP) [23]. A media server can announce clients by multicasting SAP packets containing a session description as a payload. These session descriptions inform the client about the media address together with each stream of a media, their encodings, clock frequencies and other required attributes [22]. The announcement packets are sent periodically over UDP and the server side does not know any client information. It only multicast the packets which can be captured by any machine registered to the same multicast address.

3.4 RFC 3984: RTP Payload Format for H.264 Video

The H.264 video codec has a very wide range of applications including low bit-rate Internet streaming applications, HDTV broadcast and Digital Cinema applications. Compared to the current state of technology, H.264 is reported to provide bit rate saving of 50% or more [24]. The codec specification contains two conceptual layers, a video coding layer (VCL) and a network abstraction layer (NAL). The VCL contains traditional video coding steps and NAL specifies the encapsulation and transformation format that are suitable for the underlying network.

An active parameter set remains unchanged throughout the video sequence and active picture parameter set remains unchanged within a coded picture. These parameter sets are

necessary for the decoding of video sequences since they contains necessary information such as picture size, optional coding modes employed, and macroblock to slice group map. These parameter sets are transmitted reliably to the decoder side prior to the actual coded video sequence.

A NAL unit consists of a one-byte header and the payload byte string. The header indicates the type of the NAL unit, the presence of bit errors or syntax violations in the NAL unit payload, and information about the importance of the NAL unit during the decoding process. The payload follows the header immediately. The NAL Unit header has the following format:

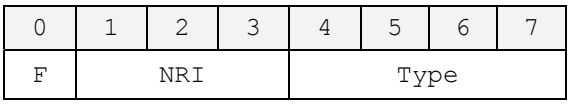


Figure 3.2 NAL Unit header structure. F is used for syntax violation.NRI with a value of 00 indicates that the content of the NAL unit is not used to reconstruct reference pictures for inter picture prediction. Values greater than 00 indicate that the decoding of the NAL unit is required to maintain the integrity of the reference pictures. Type specifies payload type.

Table 3.2 NAL Unit Types

Type	Packet	Name
0	Undefined	
1-23	NAL Unit	Single NAL Unit packet
24	STAP-A	Single-time aggregation packet w/o Decoding Order Number
25	STAP-B	Single-time aggregation packet w/ Decoding Order Number
26	MTAP16	Multi-time aggregation packet

27	MTAP24	Multi-time aggregation packet
28	FU-A	Fragmentation unit w/out Decoding Order Number
29	FU-B	Fragmentation unit w/ Decoding Order Number
30-31	Undefined	

RFC 3984 describes a new payload format for transmission of H.264 coded videos over RTP. It defines necessary packetization formats for transmission of H.264 NAL Units. Now we will evaluate these formats briefly.

Single NAL Unit Packet: This packet type must contain only one NAL unit. The structure of the single NAL unit packet is shown in Figure []. Here, the first octet is also the RTP payload header.

Aggregation Packets: Two types of aggregation packets are defined by this specification. Single-time aggregation packet (STAP) aggregates NAL units with identical NALU-time. Multi-time aggregation packet (MTAP) aggregates NAL units with different NALU-time. The RTP timestamp must be set to the earliest of the NALU times of all the NAL units to be aggregated.

Fragmentation Units: This payload type allows fragmenting a NAL unit into several RTP packets. Doing so fragmentation is achieved on the application layer instead of relying on lower layer fragmentation. RTP sequence numbers are assigned in ascending order for the fragments of the same NAL Units. By that way, no other NAL Unit fragment is sent between the first and the last fragment of a NAL Unit.

Packetization Modes

RFC 3984 specifies three cases of packetization modes:

- Single NAL unit mode
- Non-interleaved mode
- Interleaved mode

Firstly, Single NAL unit mode is used when the value of the OPTIONAL packetization-mode MIME parameter is equal to 0. It is primarily intended for low-delay applications. Only single NAL unit packets can be used in this mode. STAPs, MTAPs, and FUs cannot be used. Second mode is Non-interleaved mode which is used when the value of the OPTIONAL packetization-mode MIME parameter is equal to 1. This mode should be supported. It is also proposed for low-delay applications. Only single NAL unit packets, STAP-As, and FU-As may be used in this mode. Lastly, interleaved mode is used when the value of the OPTIONAL packetization-mode MIME parameter is equal to 2. STAP-Bs, MTAPs, FU-As, and FU-Bs are supported by that mode.

3.5 Conclusion

In this chapter, a brief overview of each related standards are given. These standards are used in order to construct an integrated and standard based streaming system consisting of a media server and media players located in the client side. They are also most popular standards used currently in other mono streaming applications. In our end-to-end stereo streaming system, SAP is used to announce available media files located on the server to the clients. Server is never aware of the existence of any client. It only sends the media file descriptions using SDP embedded inside SAP packets. The recipient of these SAP packets can request the corresponding media from media server after session setup. For session

initialization between client and server RTSP protocol is used. The assigned address and ports by the client side are sent to server side over TCP using RTSP SETUP request. After RTSP session initialization, the media file is streamed using a RTP/UDP. The media files are packetized/depacketized based on the RTP Payload format for H.264.

Chapter 4

SYSTEM ARCHITECTURE AND IMPLEMENTATION DETAILS

4.1 Architecture Decisions

The system consists of two components, client and server. As a client, an open source media player called *VideoLAN client* is modified to handle stereo video. Server was implemented as an RTSP based multi-client server using a well known, open source RTP/RTCP [20] stack for message structuring at the packet level.

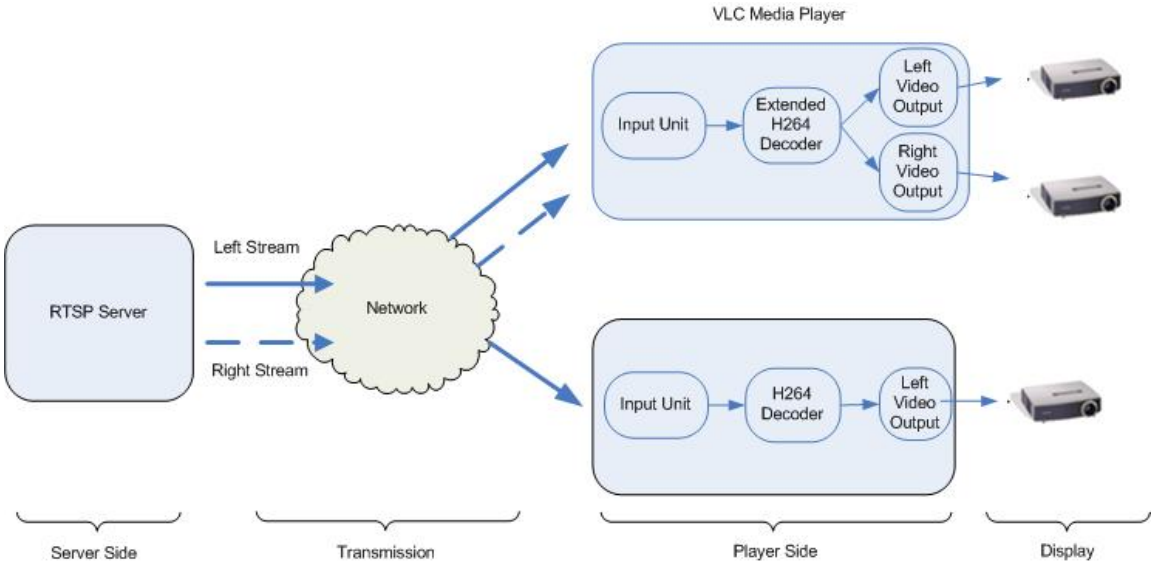


Figure 4.1 End-to-End Stereoscopic Streaming System

VideoLAN project offers a complete solution for video streaming and playback. As a client, it is a highly portable multi-media player supporting various video and audio formats [26]. Although it supports video streaming when used as a server, we used it only as a player in our system. It is well known that an efficient client side implementation of a high-quality real-time streaming system is very hard. So, instead of developing the client side from scratch, we chose to extend an already working client. Streamed data should be processed in real-time fast enough without dropping frames on the client side. If client's video decoding and data processing speeds are too slow, the received video frames are dropped before display. We can briefly explain the situation as follows: in real-time applications each video frame has decoding and presentation times. If a frame decoding time passes the local wall-clock time of the client machine, or its presentation time is too far from the current time of the local machine, player system drops the frame before or after decoding depending on the circumstances. This results in a broken video sequence. We can state that an efficient implementation is crucial for player performance. Here, VideoLAN project offers a good solution as a client. It has high performance of video processing as a full-featured cross-platform media player. VideoLAN's programming language also affects its efficiency. C achieves significant performance gain during processing. In our case, stereo video processing puts additional load on the player. Time consumed for mono video processing is doubled on the player side, since the number of displayed video frames processed by decoder or any other module such as demultiplexer or display unit is twice that of a mono video sequences with the same frame rate.

Among the existing open source video streaming platforms, I investigated Darwin Streaming Server [30], GPAC [31] and VideoLAN Client/Server [26]. Apple QuickTime Streaming Server (QSS) and its open source version Darwin Streaming Server (DSS) supports streaming of H.264 [34] coded video wrapped inside MPEG-4 [33] or 3GPP [35] file formats across the Internet using RTSP and RTP protocols[20,21]. In order to stream

video, these systems need special information about the media files. This information is carried in a track called hinted track and hinted tracks are only used when streaming media over RTP. Darwin also works with VideoLAN Client used as player. I verified that Darwin can stream raw H.264 media files over RTP and RTSP to VLC. Another project called GPAC, which is developed as a multimedia framework based on MPEG-4 standard, also supports streaming H.264 coded media files stored inside MPEG-4 file format [33]. Both of these two projects are currently being developed with latest popular standards. Their mono streaming features have been difficult to modify for our stereo streamer. Therefore, I decided not to use these systems considering that the total amount of time needed for source code investigation and for modification of the code for our stereo case would be too long.

In addition to these two platforms, VideoLAN project has two solutions for the server side. One of them is VideoLAN Server (VLS) which can stream MPEG-1, MPEG-2 and MPEG-4 files, DVDs, digital satellite channels, digital terrestial television channels and live videos on the network in unicast or multicast. The other solution is VideoLAN Client (VLC), which can be used both as a server and a client. It also supports streaming of MPEG-1, MPEG-2 and MPEG-4 files, DVDs and live videos on the network in unicast or multicast. Although as a player I deeply investigated the implementation of VideoLAN source code to be used as a client, both of these alternatives are not used. The main reason is their streaming features. Both servers support H.264 video streaming when encapsulated in MPEG-TS file format [38] over RTP. They can not stream them over RTP by itself using any payload formats other than MPEG-TS. I decided to implement my own server from scratch. This decision took shorter for me and gave me full control over the server side source code. I implemented the server based on the standards used in these well-known commercial and open source mono streaming platforms.

4.2 Server Side

Server side application is implemented as an RTSP server. The server is designed as a multi-client server. The main purpose of the server implementation is to hold and manage the H.264 coded media files and to serve them in real time to connected clients upon request. For RTP and RTSP message structures and constructions, two well known open source library was used: JRTPLIB [28] RTP stack and Vovida RTSP stack [29]. First of all, the architecture and design of the server side will be discussed in detail. Then, information about the open source libraries employed will be given in the consecutive sections.

4.2.1 Architecture of the Server Side

The client and the server need a mechanism in order to inform each other before media transmission. For that purpose one of the service discovery methods is required. In this application, Session Announcement Protocol (SAP) is used for service discovery. After the server is launched, server starts to send announcement packets periodically for each media file in its play list. As long as it runs, these announcements are multicast to a pre-selected address to inform clients about available video files. Clients can learn corresponding RTSP URLs of the media files by listening to this multicast address. This address can be any one of the addresses specified in RFC 2974. IPv4 global scope sessions use multicast addresses in the range 224.2.128.0 - 224.2.255.255 with SAP announcements being sent to 224.2.127.254 (note that 224.2.127.255 is used by the obsolete SAPv0 and must not be used). Moreover SAP announcements must be sent on port 9875 [23]. Consequently, multicast address 224.2.127.254 and port 9875 are used for SAP announcements.

In addition to notifying the clients, SAP announcements are also used to publish media and session descriptions to the clients interested in the corresponding announcement. For

each media file on the media server an announcement is broadcasted. Whenever a client receives a SAP announcement, it also receives the session description of the corresponding announcement. The session description of a sample media file carried inside the SAP packet is as follows:

```
v=0
o=- 2890844526 2890842807 IN IP4 172.17.4.143
s=RTSP Session
t=0 0
m=video 0 RTP/AVP 96
a=rtpmap:96 H264/90000
a=tool: Multiview Streamer
a=control:rtsp://172.17.4.143:6500/balloons
a=view:stereo
```

When client *C* receives the SAP announcement packet, it also learns the media description. The media description contains descriptions of the presentation and all its streams, including the codecs that are available, dynamic RTP payload types, the protocol stack, and content information such as language or copyright restrictions, etc. The detailed format of the SDP can be found in RFC 2327. Since we streamed H.264 coded stereo video files, video codec is defined as H.264. As a streaming protocol RTP is used. The media file is streamed using RTP payload format for H.264 video. For this payload format, the payload type is 96 and the video files are coded with 90000 kHz clock. These properties are defined under `m` and `a=rtpmap` fields.

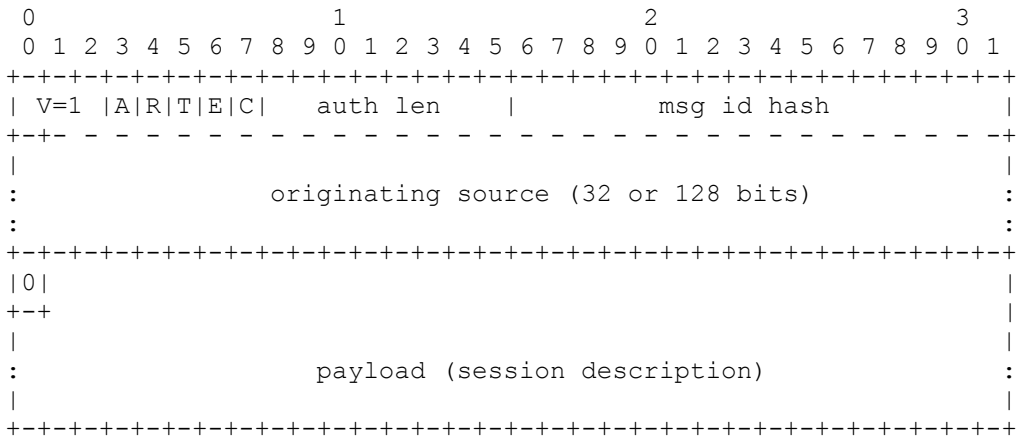
For our application, we have to identify a stereo video in the media description. The client must know that the video received is either a mono video or a stereo video. The `attribute` mechanism is the primary way for extending SDP and tailoring it to particular applications or media. There are lots of attributes which classify the streamed video. But there is nothing which denotes a stereo video. I added a new attribute called `view` to the

session description protocol which shows whether a video is stereo or not for our stereoscopic streaming system. This attributes is as follows:

```
a=view:stereo
```

Where `view` attribute shows that the video served by server upon request is a stereo typed video. There are some other ways to create a session description for a stereo video. One of them is to specify two steams under separate `m` fields. We can write two `m` fields for each media streams one for left view and one for right view. However we have a single file and therefore a single RTSP URL for that file is announced. Thus, I decided to define a new attribute for stereo videos. The media announcement is done as a single media stream, because of the single RTSP URL for a stereo video located on the server side. By the way of the `view` attribute the client C knows whether the streamed video file is coded as stereo or not.

The SAP packets are broadcasted over UDP. The packet format of SAP in our application is as follows:



where optional fields are not used. First 32 bits of the packet are used as SAP header. After that, session description is inserted as a packet payload. In the application, the SAP headers are coded as:

The first byte of the header is set to `0x20(00100000 = VVVARTEC`):

`V= Version number is 1. The version number must set to 1 according to [2].`

`A= Address type is 0. So it shows our originating source field contains a 32-bit IPv4 address.`

`R= This field is reserved. SAP announcers must set this to 0.`

`T= This field indicates the message Type. This is a session announcement packet so it sets to 0.`

`E= Encryption bit is set to 0 in our application.`

`C= Compression is not used, so set the bit to 0.`

The second byte of the header is set to 0x00 (00000000 = auth len). It is set to zero so no authentication header is present.

The consecutive two bytes shows the message Identifier Hash. A 16 bit quantity that, used in combination with the originating source, provides a globally unique identifier indicating the precise version of this announcement [22].

After announcements of session descriptions by SAP announcement, we need a mechanism to support streaming video on demand. Our system is a video on demand system, so it is also a real-time streaming system allowing viewing while the video is being streamed. In order to provide video on demand, the server is designed as an RTSP streaming server. The `control` attribute in session description denotes that the media can be accessed by using the RTSP protocol. As stated above, there is only one RTSP URL for a stereo video. The main reason for that is the dependency of right view to left view. The client C could not receive right view without left one, but vice versa is possible. Therefore

defining a single media file with a single RTSP address is reasonable. A client accesses to the video by a single RTSP address. Then it identify the video format by looking at `view` attribute and agree to receive only left view or both of the left and right views depending on its display capacity.

The client capturing the SAP packets parses them and gets the session description of the media file. A client can connect to a server whose address is given in the control attribute field of the session description embedded in the SAP packets.

In order to implement an RTSP server, referring to RFC 2326, the server works with SETUP, PLAY and TEARDOWN implementations. The other RTSP methods can also be added in the future depending on the requirements. However currently, the server works with minimum functionality. However, the design and implementation provides future extensions. The main purpose of RTSP usage is to have a method between client and server to initiate media streaming in real-time. There are some other methods in order to initiate an RTP session. But to create a standard based application real time streaming protocol, RTSP, was the correct solution for session initiation. The streams controlled by RTSP may use RTP. But the operation of RTSP does not depend on the transport mechanism used to carry continuous media [21].

Since this is an on demand system, the client requests a video from a media server by the way of the RTSP SETUP request. After ports and other media transfer options such as codec type etc are arranged, the session starts by RTSP PLAY request of the client. The client can control media file streamed by the server.

The system uses unicast to deliver media. A sample message flow between client and server is as shown below. *C* is the client and *S* is the media server.

```
C->S: SETUP rtsp://172.17.4.143:6500/balloons RTSP/1.0
      CSeq: 1
      Transport: RTP/AVP/UDP;unicast;client_port=3056-3057
```

```
S->C: RTSP/1.0 200 OK
      CSeq: 1
      Session: 12345678
      Transport: RTP/AVP/UDP;unicast;client_port=3056-3057;
                 server_port=5000-5001


C->S: PLAY rtsp://172.17.4.143:6500/balloons RTSP/1.0
      CSeq: 2
      Session: 23456789
      Range: smpte=0:10:00-


S->C: RTSP/1.0 200 OK
      CSeq: 2
      Session: 23456789
      Range: smpte=0:10:00-0:20:00
      RTP-Info: url= rtsp://172.17.4.143:6500/balloons;
      seq=12312232;rtptime=78712811

C->S: TEARDOWN rtsp://172.17.4.143:6500/balloons RTSP/1.0
      CSeq: 3
      Session: 12345678

S->C: RTSP/1.0 200 OK
      CSeq: 3
```

When the server starts, a thread listening RTSP messages also starts to run in addition to the SAP announcement thread. The server listens for RTSP messages. All the RTSP messages arrived are queued and identified using message type. If the received message is a SETUP message a new session is launched. Otherwise other RTSP messages are handled depending on the current state of the session initiated before. Since there can be lots of clients connected to the server, the system needs a state machine which holds the current state of the each session.

Whenever a new SETUP request arrives, a new RTSP session is created and inserted to the `SessionMap` list. Each received messages are forward to the corresponding session after that time by using that list. The state machine handles all this forwarding and

processing after a message has arrived. Server listens RTSP packets from a port. That port is announced in the SAP packets: e.g. `rtsp://172.17.4.143:6500/balloons`. A client C sends its RTSP messages for the corresponding media file to that specific port carried in the session description. In this example client sends messages to port 6500 which server listens for new RTSP messages.

```
If it is a SETUP msg
      Create new session
      Add this session to the list ` sessionMap ` of RTSP Director
      Send arrived msg to StateMachine for processing

else if other kind of msg
      Retrieve the corresponding session from `sessionMap` by using
      sessionId
      Send arrived msg to StateMachine for processing
```

The messages are processed by the StateMachine class. Each RTSP session holds a state variable which indicates the current state of that session. State machine gets the current state of the corresponding session and processes the received messages, and then it sets the last state of that session after processing the message. The state diagram can be found in Figure 4.2.

At the beginning, all the session objects are in the INIT state whenever they are created. When a SETUP msg arrives, SETUP processes are handled and then session changes its state to the READY and waits for another msg. If the msg is PLAY, the server initiates an RTP Session for that RTSP session and transmits the media file over RTP to the corresponding client whose transport information are retrieved during the SETUP process.

Our RTSP server does not have the RECORD functionality. The state diagram of the state machine is as follows:

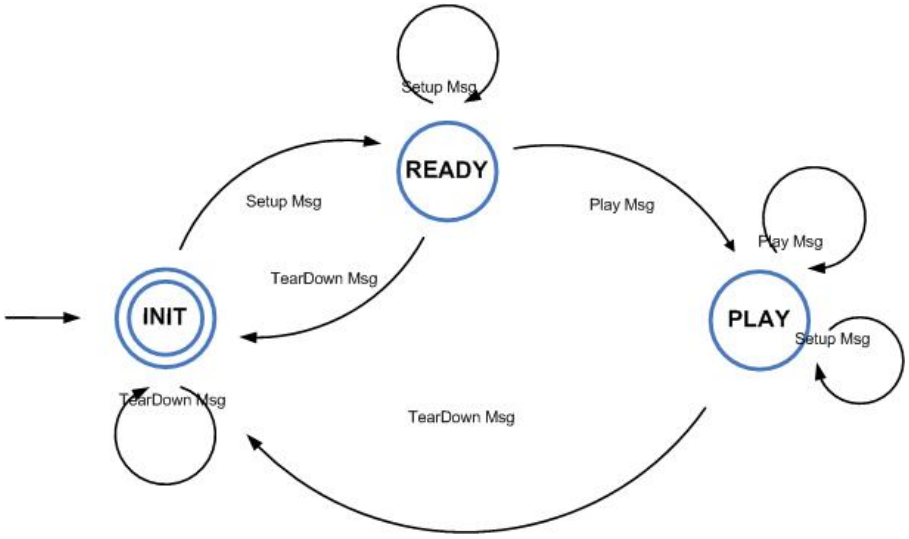


Figure 4.2 State Machine transition diagram of RTSP Server

The requests OPTIONS, DESCRIBE, GET-PARAMETERS, SET-PARAMETERS do not have any affect on the server and the client states. Moreover they are not supported by the current implementation of our software. Therefore they are not listed in the state diagram.

The client can be in any one of the following states:

- Init: SETUP has been sent, waiting for reply.
- Ready: SETUP reply is received.
- Playing: PLAY reply is received.

The RTSP messages arrived are parsed and corresponding responses are sent to the corresponding client. In the responses Connection, Content-length, Content-Type, Content-Language, Content-Encoding, Transport headers can be included by message generation functions of the Vovida Rtsp Stack library. Since our application is a RTP compliant

implementation, we can also add the RTP-info field to the message headers. By using these headers, the state machine can do the corresponding processes to setup session and play the media file. The details of the library will be explained later in this chapter.

When the state machine receives a message, it identifies the related process to be handled. If the identified process is:

SETUP Process:

- A file handler is created and set as the related file handler of the session.
- Port pairs of the client are taken from transport header of the arrived message.
- If video is stereo two RTP ports are reserved, otherwise a single RTP port is reserved from rtpProcessor class object.
- New session is created with these properties.
- New RTSP SETUP response is created and sent to the client.

PLAY Process:

- New RTSP PLAY response is created and sent to the client.
- The generated file handler in SETUP process is called and its URL field is set to the RTP-Info header.
- New RTP Session for media streaming is launched with arranged specifications.

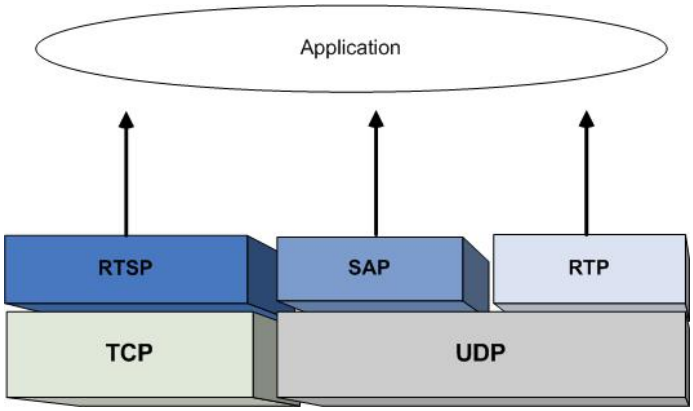


Figure 4.3 Protocol Layers

The main focus thus far is about the procedures which should be done before media transfer. I will give the details of the transmission of stereo video over RTP from server to client next. Since our main focus is the real time applications, RTP over UDP is used as the transfer protocol. One of the problems solved is the video frame synchronization on the receiver side for stereo video frames arrived from two independent channels using RTP timestamp.

On the server side each RTSP session contains an RTP Session object and initiates that object whenever a PLAY request is received at the server side. When `createRtpSession` function of the RTSPSession object is called, the server ports, one for left channel one for right channel, are set and sessionDirector object is generated to control the RTP session between the server and the related client. SessionDirector class is added to the JRTPLIB in order to manage the stereo RTP sessions.

As mentioned before, each RTSPSession has a file which will be played upon request. When the SessionDirector object arranges all the RTP initializations for both channels, the corresponding session is added to a list which holds the session ids to which media transfer will be launched. A thread starts to run and in each run it retrieves a session from the list

and reads the corresponding H.264 file with the session to transfer. In each run `rtsprtpdirector` reads 2xn number of units from the file using FileHandler object of the related RTSPSession. We try to read 2xn number of units because one of each consecutive frame is left and the other is right.

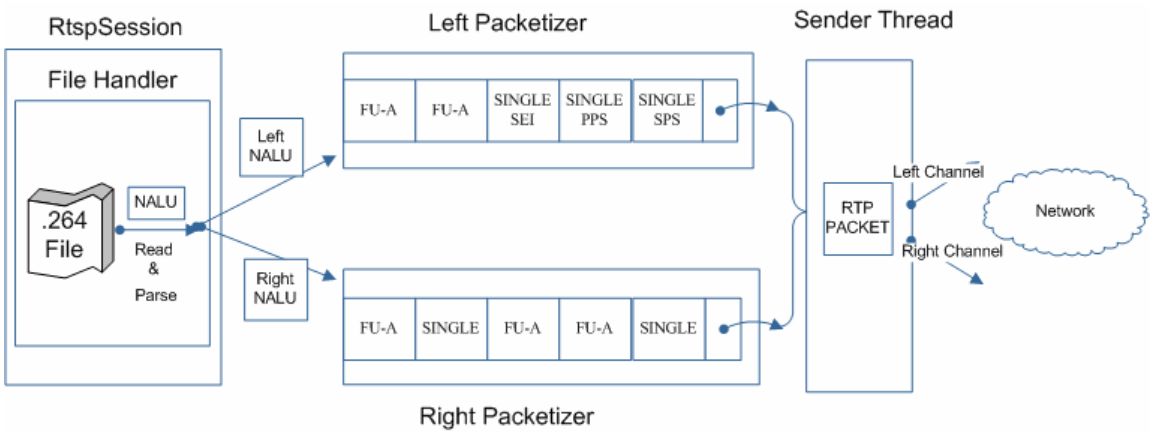


Figure 4.4 RTP Session: One RTP Session is created for requested H.264 stereo video on server side upon PLAY request of the client.

Bitstream encoded by MMRG Multiview Codec contains NAL units of both left and right views in a consecutive order. While reading from file, a parser which is specific to stereo H.264 file format (.264) is used to parse the file NALU by NALU. H.264 file format contains only NALUs consecutively without any specific file header. The file is parsed using NALU types. NALUs with type 24 are packetized as the right channel packets and all other NALUs are packetized as the left channel packet. The main point of placing all not 24 type NALU to left channel is to support mono vision to the client. Left channel is the mandatory channel to be established between server and client. However, right frames are coded with reference to left frames and right channel will be displayed only by stereo video capable clients. Each NALU is placed in their corresponding buffer, one for left NALUs

and one for right NALUs. The NALUs need to be packetized before transmitted over the network. Therefore in this application the NALUs, which are coded as one frame per NALU for our case, are put to the buffers after packetized. The server side RTP packet packetization is implemented based on the RTP payload format for H.264 video [24]. Three packetization modes are defined in this payload format. I implemented Single NAL Unit mode and Non-interleaved mode which are intended for low-delay applications. Single NAL Unit Mode allows transmission of a single NAL unit and Non-Interleaved Mode uses single NAL unit packets, STAP-As(Single-time Aggregation packet without Decoding Order Number) and FU-As (Fragmentation Unit without Decoding Order Number) [24]. I used FU-As packetization structure to transfer NALUs with size exceeding the network MTU. So NALUs are fragmented using FU-A packetization on the application layer instead of relying on the IP layer fragmentation. Other packets with smaller sizes are packetized as Single NAL unit packets. The related packet formats can be found in [24].

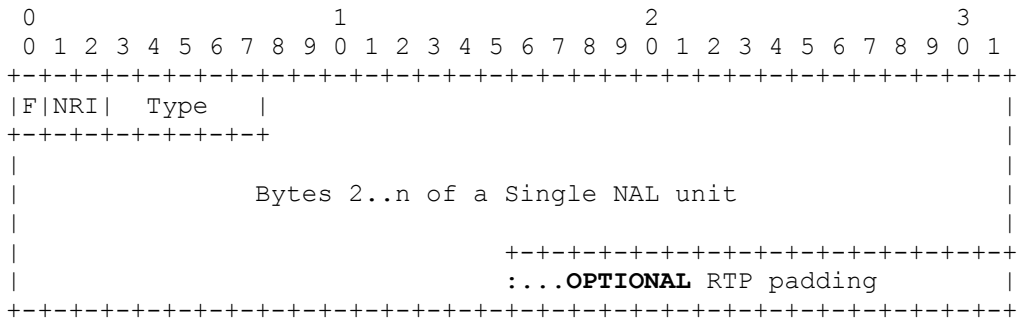


Figure 4.5 RTP payload format for single NAL unit packet

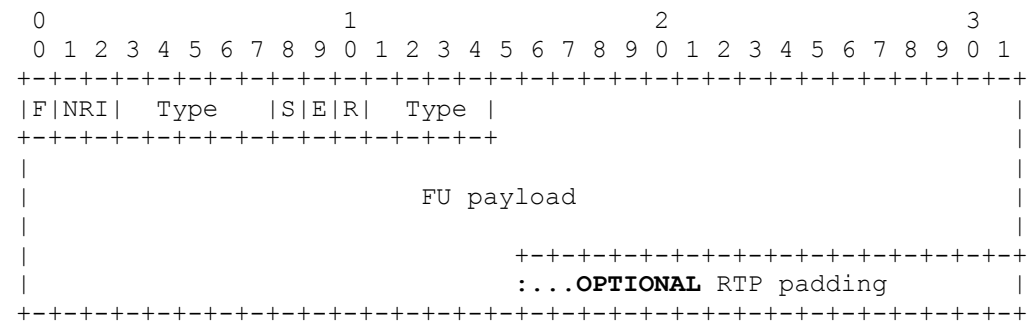
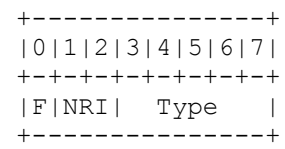


Figure 4.6 RTP payload format for FU-A

The syntax and semantics of the NAL unit type octet are specified in [24], but the essential properties of the NAL unit type octet are summarized below. The NAL unit type octet has the following format:



The semantics of the components of the NAL unit type octet, as specified in the H.264 specification, are described briefly below.

F: forbidden_zero_bit. The H.264 specification declares a value of 1 as a syntax violation.

NRI: nal_ref_idc. A value of 00 indicates that the content of the NAL unit is not used to reconstruct reference pictures for inter picture prediction. Values greater than 00 indicate that the decoding of the NAL unit is required to maintain the integrity of the reference pictures.

Type: nal_unit_type. This component specifies the NAL unit payload type.

In both of these packetization modes, the transmission order of the RTP packets shown by the sequence numbers is taken as the decoding order of the NAL units. Since the encoder does not support B frames coding, packet structures which do not contain decoding order numbers are usable in this application. The timestamp carried on the RTP header are used both to arrange the decoding order of the frames and to evaluate presentation time of frames. In addition to that, RTP timestamps are used to synchronize the frames. The client application arranges the play-out time by using the relative order of the frames positioned by the RTP timestamps. Since we stream two video files, we set related frames to the same timestamp supposing same sampling rate for videos derived from the sender`s 90 kHz clock.

For inter-media synchronization, lip-sync strategy between audio and video samples are investigated. Since we have the same kind of media type, two video files which are sampled with the same clock, we do not need to use lip-sync between media files. In this application, the coded videos are 25fps. Therefore a 3600 sample unit timestamp increment is applied for each video frame with 90 kHz clock. The initial timestamp value for the first frame is a 32 bit random number. Since I use the relative timestamp difference in order to evaluate the presentation and decoding time of the frames, a random initial value can be used. The packets which are the part of the same NALU have the same RTP timestamp referring to payload format [24]. In the server side application a 100 sample unit timestamp difference is applied between related left and right frames in order to sort the left and right frames at the client decoder queue. Since we use a single decoder which firstly decodes left frame and then decodes the corresponding right frame, correct order of the left and right frames in the decoder queue is essential. The 100 sample unit timestamp difference is handled by making the presentation times equal on the display module of the client application after decoding to display corresponding left and right frames at the same time.

In addition to these, the H.264 parameter sets are fundamental parts for video coding. Receiver must receive them before the decoding process. So we transfer them to the receiver side prior to the actual RTP sessions. These parameter sets are the first 3 NALU of the video files and they are sent over left channel since left channel is the main channel to be transferred. Currently, parameter sets are sent over RTP/UDP prior to the actual video data. However, it can be added as an attribute to session description carried inside SAP packets in order to guarantee their arrival to receiver side.

4.2.2 RTP Stack: JRTPLIB

In our application, JRTPLIB version 3.3.0 which is an object-oriented library written in C++ is used for RTP support. Library helps me in using the Real-time Transport Protocol as described in RFC 3550.

The library makes it possible to send and receive data using RTP, without worrying about SSRC collisions, scheduling and transmitting RTCP data etc. The user only needs to provide the library with the payload data to be sent. The library supplies several classes to create an RTP application. We need just RTPSession class for building an application. This class provides the necessary functions for sending RTP data and handles RTCP part internally. Library documentation and full API can be found at [28]. Library uses pthreads for thread structures at its basis. It uses JThread library which contains wrapper functions for pthreads functions to make the threading usage easy inside the library.

As I stated before, this library is used for RTP packet formation. The Packetizer class fills the buffers with the packets which obey H.264 payload format. These packets are transmitted inside the RTP packet payload field. Required RTP packet formation is achieved by using JRTPLIB which provides a fully compliant RTP packetization as in RFC 3550. Each RTP Session class in our server side application corresponds to two

RTPSession of the library one for left channel and one for right channel. Sessions are independent from each other. To manage stereo video transmission, a new class called SessionDirector is added to the library. This class provides users to create a director which manages two packetizers and RTP Sessions.

4.2.3 RTSP Stack: Vovida RTSP Stack

The Real Time Streaming Protocol, or RTSP, is an application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions.

This RTSP stack is fully compliant with RFC 2326 (Real Time Streaming Protocol, RTSP) [29]. It is written in C++ and supports TCP protocol but not UDP protocol. The RTSP code release mainly contains two portions:

- rtspstack - Contains the source code that implements the RTSP protocol
- rtspif - Contains the source code for RTSP interface. It is a wrapper of RTSP stack

4.2.4 Graphical User Interface

Server side has a simple user interface which controls the corresponding server functionality. For implementation, QT [12] is used on Linux platform. It includes C++ class libraries; it is a cross-platform gui software toolkit.

On startup, some announcements which have remained from the last server run are read from a file named *media.txt*. This file holds all active announcements from the last server run. Server also appends definition and path of each new file to be announced to this file in order not to lose the file paths on each startup.

A file is specified with its media identifier and its path on the local machine. User can add a new H.264 file by selecting a file using *Browse Button*. This button opens an explorer to find an H.264 file from the disk. The selected file extension should be ".264" otherwise selected file is not accepted. Files are identified by their path, but user should also enter a definition which will be announced in the RTSP URL of the file as the textual media identifier.

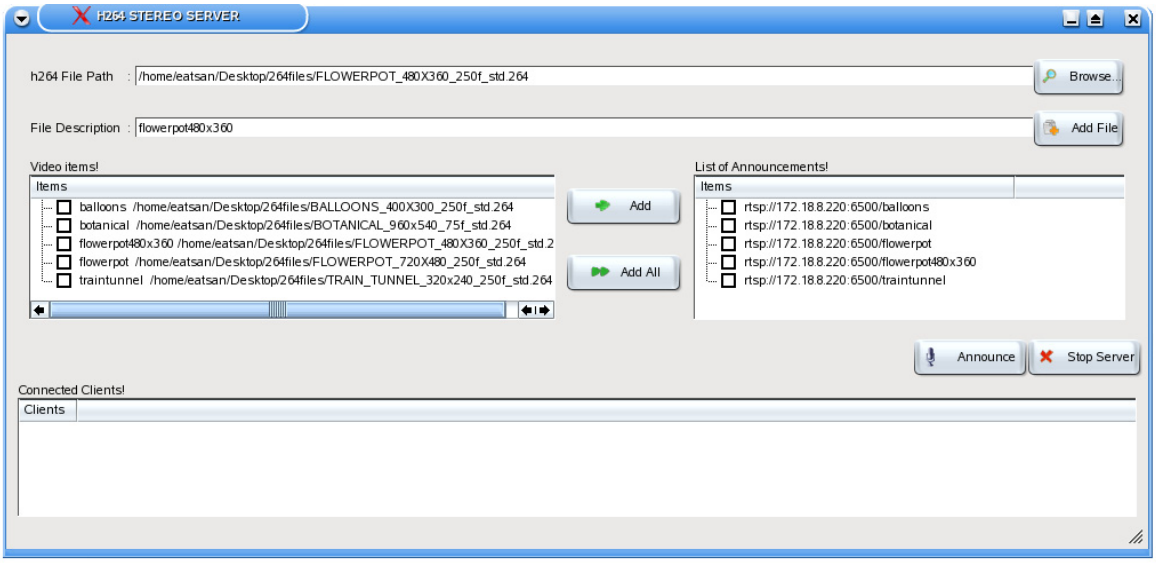


Figure 4.7 A Snapshot of RTSP Server GUI.

User can select the files to be announced form the list. Whenever user clicks to add the selected files to the announcement ListBox interface component, the corresponding announcement structures are created according to media file identifier and server properties

and then added to the announcement list. This structure contains session description also including RTSP URL as mentioned before.

After this, the user can start the announcement thread to broadcast the session descriptions held in the announcement list and the RTSP server thread by clicking *Announce Button*. Application sends announcement packets and starts listening received RTSP messages from the clients on port 6500. The connected client properties an also be viewed from the user interface.

4.3 Client Side

To be used as a player, open source software VideoLAN Client (VLC) is modified for our purpose. VLC is a highly portable multimedia player for various audio and video formats (MPEG-1, MPEG-2, MPEG-4, DivX, mp3, ogg,...) as well as DVDs, VCDs, and various streaming protocols [26] Although, the VLC player can play raw h.264 bitstream from the local disk, VLC system does not support raw H.264 streaming over RTP. Currently, VLC only supports streaming of MPEG-TS file format over RTP. Thus, we used VLC as a player and modified its stream receiver.

4.3.1 Architecture of VideoLAN Client

It is designed to stream MPEG videos on high bandwidth networks. VideoLAN was originally designed for network streaming but VideoLAN's main software, VLC media player has evolved to become a full-featured cross-platform media player.

VLC is written around LibVLC. It is a full featured multimedia player developed as open source software. In addition to core VLC, player increases its module functionality by using third party libraries. Among these libraries there are lots of audio/video codecs such

as libmad an MPEG audio decoder, libmpeg2 an MPEG1/2 video decoder, ffmpeg an extensive audio/video codec library supporting many formats such as MPEG4, H263, H.264, WMV/A etc., libtheora which is a video decoder for Theora codec, a lossy video compression method derived from On2's VP3 Codec. There are also GUI framework libraries such as wxWidgets, KDE, GTK+1/2. In addition to these categories there are lots of miscellaneous libraries such as liveMedia used for multimedia streaming(RTP/RTSP, RTSP,SIP) library, libdvdread for reading DVD-Video images, matroska support for a new format, libopenslp an open source implementation of Service Location Protocol.

LibVLC

LibVLC is the core part of the VLC media player. It is a library providing basic functionalities required by a media player such as: stream access, de-multiplexing, audio and video output, and plug-in handling. The source code of core VLC can be found under *src/* directory. This directory contains subdirectories each of which supporting a different functionality:

- src/audio_output: initializes the audio mixer, finds the right playing frequency and then resample audio frames received from the decoder(s). It contains functions for audio output API towards decoders for audio output filters management and internal management of input streams for the audio output.

- src/input: opens an input module, reads packets from a source such as a local file or a stream, parses and demultiplexs them if necessary and passes reconstituted elementary streams to the decoder(s).

- src/interface: contains code to provide interface access for other threads and CD/DVD-ROM ejection handling functions.

- src/misc: contains miscellaneous functions used by other units of the VLC such as data block management functions, high resolution time management functions, network management functions, object and thread handling functions.

- src/playlist: contains playlist related code such as Playlist thread, playlist services_discovery module and related management functions.

- src/stream_output: contains stream related functions such as muxing, and packetizing a stream, also contains announcement handling functions. This part has not been competed yet.

- src/video_output: initializes the video display unit, retrieves ready pictures from decoder, calls related display module and organizes displaying and rendering at the corresponding display time. This module describes the programming interface for video output threads. It includes functions allowing opening a new thread, sending pictures to a thread, and destroying a previously opened video output thread. It also contains appropriate picture management functions.

Module Structure

Modules are located under *modules/* subdirectory. They are loaded at run-time. There are lots of different modules each functioning for a specific service support. Every module may offer different features that will be best for a particular environment, a particular file

type, or a particular stream. There are two module structures: plug-in modules and built-in modules. Plug-in modules are loaded dynamically. You can use existing modules currently added to VLC or add a new module for VLC. The detailed information can be found in [1]. However, built-in modules can be built directly into the application using core VLC. If your system does not support dynamically loadable codes such as plug-ins, developing built-ins will be more appropriate. Built-in and plug-in module management functions can be found under src/misc/modules.c

On startup, VLC creates a bank of all plug-ins and built-ins. Every plug-in is checked with its capabilities which identify it as a plug-in for demuxing, accessing, coding and etc. Whenever a module is needed, *module_Need* function is called. This function returns the module that best fits the asked capabilities for demuxer, video-decoder, audio-decoder, video output etc. It parses the module list for capabilities and probes each of them to test that this module can do what we need. Then, it returns the first successful module defined by *module_t* structure, which is the module description structure.

Thread Structure

VLC is a multi-threaded application. It handles functions of each module via a different thread. Playlist unit, input unit, decoding unit and display units are all separate threads. VLC thread structure is modeled on pthreads [41]. However, it does not directly use pthread library, there are wrapper functions under *src/misc/thread.c* to facilitate the usability inside VLC code.

- vlc_threads_init : initialize thread system
- vlc_threads_end : stop thread system
- vlc_mutex_init : initialize a mutex

- vlc_mutex_destroy : destroy a mutex
- vlc_cond_init : initialize a condition
- vlc_cond_destroy : destroy a condition
- vlc_thread_set_priority : set the priority of the current thread
- vlc_thread_join : wait until a thread exits

VLC carries out decoding and playing operations asynchronously. Decoding and playing are done through distinct threads. The aim of multi-threaded design is to guarantee that playing is done at the right time without being interrupted by decoding process. In this design, output of the decoder thread is the input of the playing thread via buffers.

The output unit is supposed to play the received video frame at the right time. Time is shown by the presentation timestamp. The decoder receives this timestamp from system and labels all samples according to this value. The time variable is a 64 bit unsigned variable, *mtime_t* structure. The related high resolution time management functions can be found under *src/misc/mtime.c*. Threads can be suspended by using *mwait* function, which is used by video output thread to wait until the right time to display previously rendered picture.

VLC Multi-thread structure also provides a compliant design for our stereo and multi-view video outputs. The synchronization between multi-view videos can be provided by using the same presentation time stamp values for the corresponding video frames processed by distinct video output threads. These threads will be scheduled to play the received video frame at the right time and the corresponding frames will be displayed synchronously.

Buffer Management

Buffer structure provides a memory area which holds received packets from one unit in order to be used as inputs for another unit. The output of one thread becomes the input of the other thread. This mechanism is achieved by buffer structures used in between these threads. Buffers are implemented based on fifo data structures. The received packets are put to the end of the fifo. This implementation is changed for stereo extension. Received packets are placed to the decoder fifo depending on the timestamp values. A video packet with a smaller timestamp value is sent to the decoder earlier. Since packets are transferred using RTP over UDP, ordering is done at the application layer to ensure that the packets are sent to decoder in the correct decoding order. Otherwise, decoder can't function. Block allocation and fifo management functions can be found under *src/misc/block.c*.

Playlist Creation

The playlist is created on startup from files given in the command line. The appropriate plug-in can then add or remove files from it. Main playlist thread plays items one by one. In each run, thread calculates the next playlist item to play using *NextItem* function depending on the playlist course mode such as next, forward etc. The returned playlist item is played by calling *PlayItem* function. This function starts an input thread for the item. In addition to the items given in the command line, an activated service discovery module can also detect new playlist items announced. A service discovery module to be used is given in the command line and created on startup. Then, *playlist_ServicesDiscoveryAdd* function calls the given service discovery module for a playlist. Service discovery runs as a separate thread. Related playlist management functions can be found under *src/playlist* directory.

Video Input Layer

The idea behind the input module is to take care of packets, without knowing what is in them. It only takes a packet at a time, reads it's ID, and delivers it to the decoder at the right time indicated in the packet header.

An input thread is spawned for every file read. Indeed, input structures and decoders need to be reinitialized because the specifications of the stream may be different. As it is stated in the playlist creation part, *input_CreateThread* is called by playlist thread inside *PlayItem* function. At first, an appropriate module to read the item is searched. *Init* and *InputSourceInit* functions initialize required resources for the input. *Init* function creates an *es_out_t* structure which is the elementary stream implementation of the input thread. All the operations about elementary stream is managed over that structure afterwards. Besides, *InputSourceInit* function tries to create a demuxer module by calling *demux2_New* function. If demuxer is NULL, function creates an access module by calling *access2_New* function. If returned module is an access plugin, it handles the functionality of opening an input socket. If returned module is a demuxer, input thread is in charge of processing the network packets and demultiplexing. Input thread calls *pf_demux* function in each run which calls the selected demux module function. The demultiplexer is responsible for parsing the packet, gathering PES, and feeding decoders. Demultiplexers for standard MPEG structures (PS and TS) have already been written in VLC. In this project, since we used the RTSP protocol, livedotcom demuxer is returned as the best fitted demuxer from module bank for our case. Livedotcom demuxer calls the LIVE55 library functions which deal with the control messages over RTSP and video packets over RTP. Arrived video packets are also depacketized by library before send to the VLC code. Here, the aim of demultiplexing is to obtain a continuous elementary stream by gathering and depacketizing H.264 video packets using H.264 payload format for RTP.

Decoder

Decoder deals with the mathematical computation part of a video system. Continuous bitstream constructed by the demultiplexer is converted to an acceptable input format for display thread via decoder module. Decoder receives bitstreams from input thread and creates samples which will be shown in the output thread. Data traffic between these three modules is managed by buffer structures. Decoder gets blocks from an input thread buffer which is gathered by demultiplexer previously and put the output pictures to the picture heaps.

Elementary stream part of the input thread creates a decoder thread for the corresponding bitstream format by calling *input_DecoderNew* function. For our case, H.264 decoder is returned from the module bank to decode H.264 bitstreams. Elementary stream code sends a block via *EsOutSend* function. This function calls *input_DecoderDecode* function to insert the demultiplexed block to decoder fifo. Decoder runs a thread and empties the decoder buffer in each run. In each run, *DecoderThread* function calls *DecoderDecode* function, which decodes an audio or a video block depending on the elementary stream type. For video elementary stream, a block of video data arrives at each time period and the corresponding decoder decoding function is called by *p_dec->pf_decode_video* callback which returns a picture. The status of the returned picture is set to READY by calling *vout_DatePicture* and *vout_DisplayPicture* functions from video output unit to remove the reservation flag of a picture, which will cause it to be ready to display. The related decoder functions mentioned can be found under *src/input/decoder.c*.

In our case, since the stream is an H.264 bitstream, the decoder created is an H.264 decoder. The module bank returns FFmpeg H.264 decoder whose implementation is

located in *libavcodec/* directory of the library. The corresponding API for FFmpeg for delivering stream data to decoder is under *modules\codec\ffmpeg \video.c*. This file contains a function called *ffmpeg_NewPictBuf*, which returns an allocated picture buffer to be filled by the decoder with sampled decoder output. The returned buffer is created by function callback *p_dec->pf_vout_buffer_new* which calls *vout_new_buffer* function. *Vout_new_buffer* functions tries to create a new video output object, if one does not exist, and then calls *vout_CreatePicture* function to create a new picture structure.

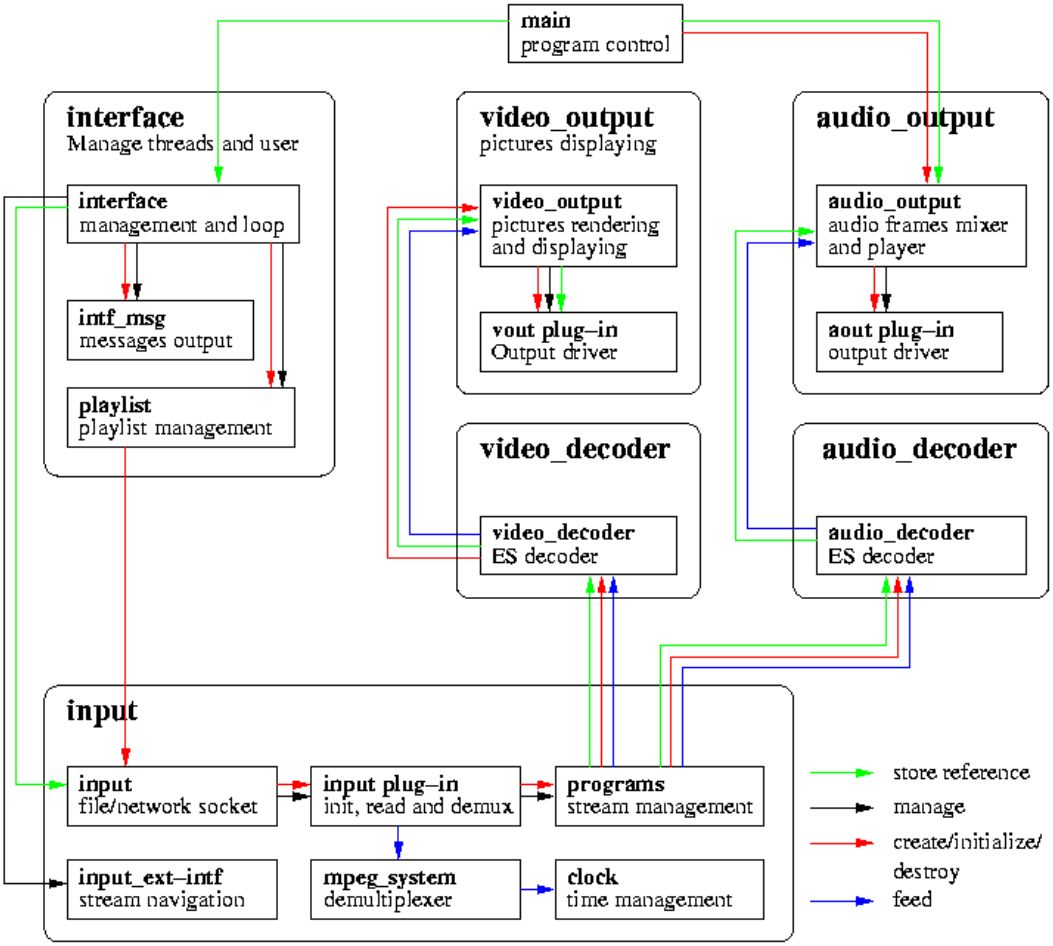


Figure 4.8 VLC Architecture [26]

Video Output Layer

All important data structures for video output layer are defined in *video_output.h* and *vlc_video.h* header files. Two main data structures of the module are *vout_thread_t* and *picture_t*. Any independent video output device, such as an X11 window or a GGI device, is represented by vout_thread_t data structure. This structure holds the current display properties, thread properties, callbacks to plug-in functions and video heaps. The heaps are filled with pictures, which are indicated by *picture_t* structure. Each picture has a status such as READY, DISPLAYED, DESTROYED and FREE etc. and a date representing exact display time. *Vout_thread_t* structure contains video picture heaps either render (to store pictures used by the decoder) or output (to store pictures displayed by the video output plug-in). The main idea here is that video output unit manages a heap of pictures. The main job of the video output thread is to handle the next picture arriving to video heap to display, find the corresponding subpicture to display (subtitles), render the picture, sleep until the specified display time of the rendered picture by using the *mwait* function and display the picture by calling the corresponding plug-in function. Video output functions can be found in *vout_thread.c* and picture management functions can be found in *vout_pictures.c* under *src/video_output* directory.

4.3.2 Stereo Extension to VLC

Thus far, VLC media player architecture and implementation details of its major units are presented. Next, I will describe the stereo extensions to VLC which support stereoscopic display of two video sequence received from the network over distinct channels. This part will also be a typical run course of VLC media player whenever an item is played.

On startup, a service discovery thread to handle announced media sessions is created besides the playlist thread creation. Although VLC supports other service discovery methods, we focused on SAP announcement method used by our server side also. The SAP module is called from the command line when vlc application starts. After thread creation, SAP thread starts listening SAP packets arriving at 224.2.127.254, which is among the specified SAP addresses in [1]. Since server announcements are broadcast to that multicast address, VLC captures the announcement packets from the server at the same multicast address and parses the received packets using SAP service discovery module. Corresponding data structures *sap_announce_t* and *services_discovery_sys_t* related to announcements are filled in here. Required modifications to the SAP module are done for stereo extension. A parser used to divide the session description carried over the received SAP packets can also identify `view` attribute. The value of the view attribute is used to identify the announced media as stereoscopic or monoscopic. The extension can be found under *modules/service_discovery/sap.c* file.

VLC adds all the announcements received from related addresses as a playlist item to its playlist. In our application, the server side announces media sessions using RTSP. Specific to stereo extension, although there are two RTP session for stereo video, they are announced only with a single RTSP URL. This is because of the dependency of the right channel to the left channel. The right channel cannot be retrieved without the left channel. Thus, announcement of two sessions with a single RTSP URL as a single media session is reasonable. As an example, VLC adds a playlist item as rtsp://172.17.4.143:6500/balloons which is retrieved from the `control` attribute of the session description.

When user calls `play` command from the GUI, the first item starts proceeding or user can call `goto` command which skips and starts from an item located in specific index of the playlist. Whenever the *PlayItem* is called for a playlist item, an item specific input is spawn. Firstly, input specifies the correct demux and network modules which can open that

item. In our application, streamer announces sessions over RTSP so all the items have an RTSP URL. Thus, VLC calls the corresponding RTSP module in order to process the received announcements. To do that, `rtsp://` field of the playlist item is parsed and *module_need* function to find a handler for the RTSP protocol is called to obtain the best module. RTSP module inside VLC is under demux module capability. It is a demux module because the received packets should be processed and de-packetized before sending them to the elementary stream part. VLC has a livedotcom API which calls the corresponding Live555 library functions for RTSP demuxing. Network operations and depacketization is handled by the Live555 Streaming Media library for H.264 bitstreams which will be presented later. As it is stated in the server part, the main reason to use RTSP is to provide a video on demand system. By that way, client can request a video from the server anytime and streaming starts upon request.

Extended VLC handles packets of left and right views over two separate channels. However a single decoder is used in order to decode the received bitstreams. For H.264 bitstream, corresponding decoder for H.264 coded data is opened by the player inside the elementary steam code. As a decoder, open source H.264 decoder implementation inside ffmpeg library is used [25], but it is modified to decode both left and right frames for stereo video. Decoder has only one buffer holding the received frames. This buffer is fed by two elementary streams processing received frames from each channel. Before sending to the decoder, the data blocks are ordered via their RTP timestamps. This ordering is also used to synchronize related left and right frames. As I stated above in the server side details, corresponding right frame timestamp is 100 sample units greater than that of the left frame, but consecutive left or right frame is 3600 sample units greater than the next left or right frame. So the received and depacketized packet, which also matches to a single NALU and also to a frame of the video file is put to the decoder buffer ordered with the timestamp values carried in the RTP headers. All left frames are decoded before the next left frame

and the corresponding right frame. Besides, all right frames are decoded before next left frame and next right frame. The decoder decodes these demultipled data blocks, produces pictures which can be rendered and sends the decoded pictures to the video output modules.

For stereo extension, there is a single decoder, but there are two video output units one for left pictures and one for right pictures connected to this decoder. Decoder forwards a frame to its corresponding video output via its frame type. Video output units are created as a child of the same decoder so they work synchronously via VLC thread structure. The video output units visualize the left and right frames in a synchronized manner by using the time information in the RTP timestamps. Both threads display the next rendered frame whenever the presentation time is reached. Since related left and right frames has the same presentation time and video output threads run in consecutive order, the frame synchronization is provided.

4.3.3 Stereo Extension to Decoder: FFMPEG Library

FFmpeg is a fast video converter and it is a third party library used as a plug-in by vlc. Most codecs were developed from scratch to ensure best performance and high code reusability. It is developed under Linux. For a detailed API of library, refer to [25].

The project is made of several components:

- *ffmpeg* is a command line tool to convert one video file format to another. It also supports grabbing and encoding in real time from a TV card.
- *ffserver* is an HTTP (RTSP is being developed) multimedia streaming server for live broadcasts.
- *ffplay* is a simple media player based on SDL and on the FFmpeg libraries.
- *libavcodec* is the library containing the codecs (both encoding and decoding).

- *libavformat* is the library containing the file format handling (mux and demux code for several formats).
- *libavutil* is a helper library containing routines common to different parts of FFmpeg.

Here, H.264 decoder under libavcodec which is modified for stereo video decoding is our focus of interest. Although streamed stereo video are encoded by the MMRG Multiview Codec, this codec is not suitable for the decoding phase, because MMRG Multi-view Codec is developed based on JM Reference Software of H.264 which does not have any concern on speed optimization. Therefore, it is not possible to decode the received stream in real-time using this codec. As a result, we improved the implementation of H.264 decoder inside ffmpeg/libavcodec library which can decode H.264 streams in real-time with the same structure of MMRG Multi-view Codec Decoder [19] in order to support stereoscopic decoding in our system.

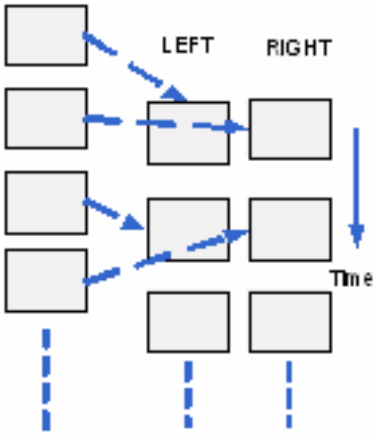


Figure 4.9 Frame Ordering of MMRG Multi-view Codec

The modified version provides us with the decoded left and right frames in a consecutive order. It is modified according to the idea of H.264 multiple reference frame structure. Decoder supposes that all frames received from the buffer are members of the same video sequence without separating them into left or right view using frame types identified at NAL unit`s start code prefix. The format of NAL units is similar, but the only difference between them is their reference list. Left frames are predicted only from other left frames. Just using left NAL Units, left video can be decoded with a standard AVC decoder. However, right frames are predicted from both left and right frames. Therefore, to decode right video, both left and right NAL Units are required. For more details please refer to *ffmpeg/libavcode/h264.c* file.

One important point here is the added prefix code to video NAL units. The sequence can be decoded providing that each NAL unit is in an acceptable format by the FFmpeg H.264 decoder implementation. It is observed that decoder divides a local 264 file into NAL units using 00 00 01 prefix at the beginning of each NAL unit. Besides, decoder implementation accepts a NAL unit, if it starts with this prefix. It identifies a NAL unit via this prefix and gets rid of this additional prefix part inside decoder after that. Since we process a stream, but not a local file, the received NAL units do not contain that prefix. Therefore NAL units with missing prefixes should be extended to an acceptable format by adding prefixes. Moreover, it is investigated that decoder throws exception if SPS, PPS and SEI type NAL units do not arrive before the decoding process. These NAL units should be gathered together and sent to the decoder as a single packet. Since these packets are received as separate NAL units over left channel; they should be formed before the fifo placement. First three packets, SSP, PPS and SEI units, are gathered together and added 00 00 01 prefix code, and other packets are added 00 00 01 prefix code. Acceptable NAL units are formed inside RTSP demuxer module, after demultiplexing is finished and before

packets are placed into the decoder fifo. The resultant NAL units are accepted by the decoder.

The sampled data by decoder are put into the allocated *picture_t* data structures. This allocation is done depending on the frame type. For left frame, picture from a left video output heap is allocated and similarly for right frame, picture from right video output heap is allocated. Thus after decoding, pictures are located in their corresponding heaps. Allocation is done by calling *vout_new_buffer* function of decoder.c file. This function is modified for stereo version of the vlc. Function receives a third parameter which identifies the allocation the function called. FFmpeg API calls that function with type -1 for right picture allocation in the second video output thread, and 1 for left picture allocation. Then the allocated picture is sent to the FFmpeg actual library to decode into. After decoding the status of the picture is set to READY by calling *vout_DatePicture* and *vout_DisplayPicture* functions from *src/video_output/vout_picture.c*.

4.3.4 Stereo Extension to Demux Module: LIVE555 Streaming Media Software

This code forms a set of C++ libraries for multimedia streaming, using open standard protocols (RTP/RTCP, RTSP, SIP). These libraries can be used to build streaming applications. The libraries can also be used to stream, receive, and process MPEG, H.264 or JPEG video, and several audio codecs. They can easily be extended to support additional (audio and/or video) codecs, and can also be used to build basic RTSP or SIP clients and servers, and have been used to add streaming support to existing media player applications, such as VLC.

For our application, Live555 library is used for RTSP support. VLC uses library as a plugin, and livedotcom support is provided via *module/demux/livedotcom.cpp* file. Library receives session desription via DESCRIBE RTSP message. However since our server does

not support DESCRIBE message and it announces media sessions via SAP, session description is retrieved from the SAP service discovery module in extended VLC. *OpenDemux* function inside the livedotcom module initially sets up the connection, the details of which are stored in the RTSP URL of the playlist item. Then a media session is created by calling *createNew* function of the *MediaSession* class. *createNew* function initializes session by using session description. Since we specified whether video is stereo or not by using `view` attribute, we extend the code for stereo from that point onward. *MediaSessio* class has a function called *initializeWithsdp*. This function parses session description and initializes session accordingly. If session description contains view attribute set to stereo, an additional subsession for the second channel is created and a flag called `*isStereo*` is set to true for the right channel. This flag will be used in order to skip the right channel during subsession iteration for RTSP protocol. We stated before that since right channel is dependent on the left channel, we need only one RTSP session between client and server for stereo video as in the mono video. The corresponding stereo properties such as ports are sent to the server via the RTSP messages after that. For details please refer to the *MediaSession.cpp* file.

After creating subsessions, a setup message by *setupMediaSubsession* function is sent to the server firstly and play message by *playMediaSession* function is sent to the server after getting the setup response. By these message exchanges, video is requested from the server side. If the requested video is of stereo type, two elementary stream structures each for one of the media subsession are created. Then, demux operation runs in a thread and gathers received H.264 packets. Live555 library has H.264 depacketization implementation for RTP payload format. The received packets are depacketized and send to decoder fifo via *StreamRead* functions, which not only adds required prefix code to the NAL units but also calls the *es_out_Send* function over previously generated elementary structure, which

locates the block to the decoder fifo. The details can be found in *modules/demux/livedotcom.c*.

4.4 Display System

The display system consists of two Sharp MB-70X DLP projectors and a silver screen as shown in Figure 4. Light from projectors are polarized using circular polarized filters. One filter polarizes light of one projector in right circular way and other filter polarizes light of the second projector in the left circular way. Both projectors projects onto a silver screen. This screen is covered with a dielectric material which keeps the polarization of light coming from projectors. The users wear glasses which guarantee to provide for left and right eye the corresponding left and right images reflected from the screen using filters matching with the projector's filters. The projectors inputs are taken from a high performance PC. The graphic card used is NVIDIA GeForce 6600, which has one VGA and one DVI output. VGA output is connected to one projector and DVI output is connected to the other projector using DVI to VGA converter. Using extended desktop feature of SUSE 10.1, which is supported and activated by the video card driver, left views from one projector is shown on the left half of the desktop and right views projected from other projector are display on the right half of the desktop. The resolution of the extended desktop is 1024x768. Each projector shows one half of the extended desktop. As a result we gain two overlap frames seen as 3D images by the users wearing polarized glasses.

4.5 Conclusion

Here, we have designed and implemented a system consisting of open source libraries. Initially, I investigated the source code of VLC media player. I gained to examine the

implementation details of such a huge open source media player. From source code, I learned the design issues of a media player including coding, demuxing, streaming, service discovery, network functionality, visualization and etc. The media player has been changed for stereo display case.

In addition to VLC; a codec library FFmpeg and a RTSP library Live555 have also been investigated. I modified and integrated all these libraries to provide a stereo media player working efficiently. While modifying the source codes, I also gained knowledge of efficient coding in C.

Similarly, I implemented a streaming server which was developed by me. For the underlying protocol functionalities, I used other open source libraries such as JRTPLIB, Vovida RTSP Stack for RTP and RTSP. Server side has been implemented using C++. As a result, an end-to-end streaming system for stereo videos has been implemented. The system has been constructed based on well-known standards.

Before design stage, mono streaming platforms were also investigated. Then I used most popular standards for our system. The main difficulty was to change VLC source code. The difficulty came from lack of VLC documentation. Therefore, using another media player could make the implementation period shorter. On the other side, the efficiency of VLC code has provided us a well-performed stereo media player.

Chapter 5

CONCLUSION

5.1 System Test

For transmission and display process, we have implemented all the modules consisting of media server and media player, and run the system with pre-encoded files. We tried the system using different videos and the video quality is found quite satisfactory comparable with a local playback. Current system is tried on 100 MB local area network with zero packet losses. Moreover, we have tried the system for multiple client connection. It has been observed that the system works fine for 5 client server connection. The tests may be done for more client connection.

The H.264 coded video increased the efficiency of bandwidth usage and this also affects the quality of the views. The tested stereo video sequences are shown in Table 5.1. Raw stereo files; *train tunnel*, *balloons* and *flowerpot*, are encoded using MMRG video codec with different sizes. We set video rate as 25 fps and we coded videos as one intra frame per 25 frame. The system gave satisfactory results for video files including videos with a size of 720x480. The quality of the video is fine on the receiver side.

The coded video sequence contains 250 frames for each view. The server side has a mode to repeat the streamed video sequences again and again. This has been added in order to test the system stability during a period. All videos have been repetitively played during 1 hour time intervals and it is observed that the streaming functionality, video continuity and 3D vision have been preserved throughout video display on the receiver side.

Table 5.1 Tested Stereo Video Sequences

Stereo Video Files	Size	Total Frames	fps	File Size(bytes)	Duration(s)	Bitrate (Kbps)
Train tunnel	320x240	250	25	1532363	10	1225.89
Train tunnel	720x576	250	25	7048910	10	5639.13
Balloons	400x300	250	25	2250141	10	1800.11
Balloons	720x480	250	25	5506666	10	4405.33
Flowerpot	480x360	250	25	2622966	10	2098.37
Flowerpot	720x480	250	25	5218491	10	4174.79

System has been implemented using C/C++ on the Linux platform. The source code has been compiled with each of the gcc 3.3.5, gcc 4.0.2 and gcc 4.1.0 versions. The server application is installed to Suse Linux 10.0 and 9.3 operating systems on computers with given features in Table 5.2.

As we stated before the media player performance is crucial for display quality. For client side, we tested the system on two computers with different features. Firstly, VLC Stereo Extension has been tested on a computer with Intel Pentium M processor and 512 MB RAM. System performance has been fine for that case. Secondly, the VLC has been installed on computer with AMD Athlon 64 Processor and 2GB ram. Although it has been mentioned that VLC source code has not been optimized for 64 machines on the developers side of the application [26], the results has been indicated that the stereo extension can also give satisfactory video quality on 64 bit machines.

Table 5.2 System Features

Software	Operation System	Processor	RAM
Extended VLC	Suse Linux 10.0	Intel(R) Pentium(R) M processor 1.60GHz	512 MB
	Suse Linux 10.1	AMD Athlon(tm) 64 Processor 3000+	2 GB
Media Server	Suse Linux 10.0	Intel(R) Pentium(R) M processor 1.60GHz	512 MB
	Suse Linux 9.3	Intel(R) Pentium(R) 4 CPU 3.20GHz	2 GB

5.2 Conclusion and Future Work

End-to-end stereoscopic streaming system has been implemented as a Media Server which can stream stored H.264 coded stereo videos to the requested client and a media player which has the processing and display feature of stereo video. The existing softwares are used whenever possible and they are extended for our design criterions. The following is the summary of the contribution of this thesis:

- Design and implementation of an end-to-end stereoscopic video streaming system using open source components with required modifications.

In design stage, available commercial and open source multimedia systems have been investigated in terms of container formats, video compression techniques, transmission protocols and available multimedia server/players. These evaluations have been done in

order to clarify whether we can extent these tools and standards for our stereoscopic streaming system.

We can state that an efficient implementation is crucial for player performance. Among the existing open source video streaming platforms, VideoLAN project offers a good solution as a media player. It has high performance of video processing as a full-featured cross-platform media player. Additionally, its thread structure proposes a simultaneous playing and decoding. In our case, stereo video processing puts additional load on the player. Therefore, these performance issues become more and more important. VideoLAN Stereo Extension has resulted satisfactory results for stereo view. However, we decided not to use existing systems for server side considering that the total amount of time needed for source code investigation and for modification of the code for our stereo case would be too long. I decided to implement my own server from scratch. This decision took shorter for me and gave me full control over the server side source code. I implemented the server based on the standards such as RTSP, RTP, SDP and SAP used in these well-known commercial and open source mono streaming platforms.

The idea behind the stereoscopic streaming system is the usage of two separate channels dedicated to each view one for left and one for right eye. Receiver can view the content of the video built from two channels as stereo depending on its display equipment and bandwidth capabilities. The resultant system proposes a video-on demand system with end-to-end streaming and 3D display mechanism.

The video on demand functionalities are implemented by using RTSP protocol. Client can request a media file announced by Session Announcement Protocol used as a service discovery mechanism of the system. Media server is implemented as a minimal RTSP Server. Server can supply multiple clients with stereo video. It generates an RTSP Session for each connected client to hold the state of the connection and the media stream. VOVIDA RTSP Stack is used on the server side to provide RFC 2326 compatible RTSP

functionality. Similarly, Live555 library, which can be integrated as a plug-in for VideoLAN Client, is used for RTSP functionality on the client side. It has been extended for our stereoscopic system. Media session within a library opens another connection as a right channel whenever stereo attribute is defined in the session description.

As a video compression standard, H.264 video compression technique is used. The video sequences for stereo view are pre-encoded as a single file by MMRG video codec. The compressed stereo video files are parsed and streamed over two different channels. The H.264 media files are packetized and depacketized based on the RTP Payload format for H.264, RFC 3984 before streaming. On the receiver side, H.264 decoder inside FFMPEG library is extended to a stereo decoding mode compatible for the decoding of our streamed video files. The corresponding decoder is integrated to the media player. In the system, the decoder is fed by two input buffers, each of which holds depaketized video data each arrived from a dedicated channel for a different view. The output picture of the decoder is send to one of two different video-output units depending on its view type, left or right. As a result, the multithreaded architecture of the VideoLAN Client provides simultaneous playing and decoding.

Stereo video synchronization is achieved by using the RTP timestamp mechanisms. Each matching left and right frames at an instant of time are assigned to the same RTP timestamp which is evaluated as 25 fps. These video frames are displayed simultaneously, when the estimated presentation time of the frames is arrived on the receiver side, *Extended VideoLAN Client*.

An end-to-end stereoscopic streaming system, which can be extended to the multi-view streaming system in the future, is constructed by the integration of the developed media server and media player as defined above. During test period, the media player performance has been satisfactory, and comparable with a local playback. Also our display equipment and setup have given acceptable 3D vision for the viewers.

As a result, we have designed a complete platform consisting of media server and compatible media player performing stereo-view capturing, real-time transmission and displaying with existing equipments. In this thesis, we achieved the transmission of pre-encoded compressed stereo video sequences over RTP and display of retrieved video after decompression using a special stereo decoder. The real-time capturing and encoding will be the next step through a complete stereoscopic real-time streaming system. Then, this system will be easily adapted to the selective transmission of true stereoscopic video sequence depending on instant user perspective for multi-view extension. Additions for used standards such as RTP, RTSP, SDP will also be proposed for multi-view case.

Currently system supports only streaming of H.264 media file format. The player functionality and integrity also increases the usage of system with the future improvements on different file formats and codec standards. Moreover, object oriented design of server side also supports addition of new file handlers for other file formats added one day.

Additionally, we will try to improve coding efficiency to reduce the bandwidth requirements in the future. Loss resilience techniques and loss concealment techniques will be added to extend the system for real Internet use with considerable packet losses.

BIBLIOGRAPHY

[1] G. Blakowski and R. Steinmetz.A Media Synchronization Survey: Reference Model, Specification, and Case Studies. IEEE Journal on Selected Areas in Communication, VOL. 14, NO. 1, January1996.

[2] Y.Chu, S.Rao, and H.Zhang.A Case for End System Multicast. Proc. of ACM Sigmetrics, June 2000.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. ACM SIGCOMM, 2002.

[4] J.G. Apostolopoulos, W. Tan, and S. J. Wee. Video Streaming: Concepts, Algorithms and Systems. Hewlett-Packard Company, September, 2002.

[5] D. Wu, Y. T. Hou, W. Zhu, Y. Zhang,and J.M. Peha. Streaming Video over the Internet:Approaches and Directions. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11, No. 3, March 2001.

[6] A. Smolic and H. Kimata. Application and Requirements for 3DAV. ISO/IEC JTC1/SC29/WG11 N5877, July 2003.

[7] Survey of Algorithms used for Multi-view Video Coding (MVC). ISO/IEC JTC1/SC29/WG11 MPEG2005/N6909, January 2005.

[8] B. L. Tseng and D. Anastassiou. Multi-Viewpoint Video Coding with MPEG-2 Compatibility. IEEE Transations on Circuits and Systems for Video Technology, 6(4):414—419, 1996.

[9] 3D Coding Techniques Draft. 3DTV,2005.

[10] B. Haskell, A. Puri and A. Netrevali. Digital Video: An Introduction To Mpeg-2. Chapman & Hall,1997

[11] [11] S. Chien, S. Yu, L. Ding, Y. Huang and L. Chen. Efficient Stereo Video Coding System for Immersive Teleconference with Two-Stage Hybrid Disparity Estimation Algorithm. ICIP, September 2003.

[12] http://www.trolltech.com/products/qt

[13] W. Matusik and H. Pfister. 3D TV: A Scalable System for Real-Time Acquisition, Transmission, and Autostereoscopic Display of Dynamic Scenec. Proc. ACM SIGGRAPPH, 2004, pp.814-824.

[14] E. Laboray, S. Würmlin and M. Gross. Real-Time Streaming of Point-Based 3D Video. Proceedings of the IEEE Virtual Reality 2004 Conference, pp. 91-98.

[15] C. Fehn, R. de la Barre and S. Pastoor. Interactive 3-DTV-Concepts and Key Technologies. Proceedings of the IEEE, March 2006.

[16] S. Hu, A Case for 3D Streaming on Peer-to-Peer Networks. Proceedings of the eleventh international conference on 3D web technology, 2006.

[17] C. Fehn, P. Kauff, M. Op de Beeck, F. Ernst, W. IJsselsteijn, M. Pollefeys, L. Van Gool, E. Ofek and I. Sexton. An Evolutionary and Optimised Approach on 3D-TV. In Proceedings of International Broadcast Conference, pages 357-365, Amsterdam, The Netherlands, September 2002.

[18] U. Fecker and A. Kaup. H.264/AVC-Compatible Coding Of Dynamic Light Fields Using Transposed Picture Ordering. EUSIPCO 2005, September 2005.

[19] C. Bilen, A. Aksay and G. Bozdagi Akar. A Multi-view Video Codec Based on H.264. IEEE ICIP 2006, October 2006.

[20] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[21] H. Schulzrinne, A. Rao and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, April 1998.

[22] M. Handley, V. Jacobson. SDP: Session Description Protocol. RFC 2327, April 1998.

[23] M. Handley, C. Perkins and E. Whelan. Session Announcement Protocol. RFC 2974, October 2000.

[24] S.Wenger, M.M. Hannuksela, T. Stockhemmer, M. Westerlund and D. Singer. RTP Payload Format for H.264 Video. RFC 3984, February 2005.

[25] http://ffmpeg.sourceforge.net/

[26] http://www.videolan.org/vlc/

[27] www.live555.com/

[28] http://research.edm.luc.ac.be/jori/jrtplib/jrtplib.html

[29] http://www.vovida.org/

[30] http://www.apple.com/

[31] http://gpac.sourceforge.net/

[32] A. Luthra, G.J. Sullivan , and T. Wiegand(eds.). Special Issue on H.264/AVC. IEEE Transactions on Circuits and Systems on Video Technology, July 2003.

[33] ISO/IEC International Standard 14496 (MPEG-4). Information technology - Coding of audio-visual objects. January 2000.

[34] ITU-T ISO/IEC 14496-10. Recommendation H.264: Advanced video coding for generic audiovisual services. May 2003

[35] http://www.3gpp.org/

[36] http://iphome.hhi.de/suehring/tml

[37] S. Wenger. H.264/AVC Over IP. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, July 2003.

[38] C.E. Holborow. MPEG-2 Systems: A Standard Packet Multiplex Format for Cable Digital Services. Society of Cable Television Engineers Conference on Emerging Technologies, January, 1994.

[39] https://www.helixcommunity.org/

[40] http://www.sun.com/

[41] IEEE POSIX 1003.1c Standard, 1995.