# UNIVERSITY OF KALMAR

# Real Time Video Transmission For Scalable Video over the Internet

**A THESIS**

**SUBMITTED TO THE DEPARTMENT OF TECHNOLOGY**

**UNIVERSITY OF KALMAR**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS**

**for the degree**

## MASTER OF SCIENCE

## FIELD OF ELECTRICAL ENGINEERING

## BY


## AMIT B. BORLIKAR

## KALMAR, SWEDEN

# ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, PD Dr. Raouf Hamzaoui for his guidance and friendship. Without his guidance, this thesis would not have been possible. Dr. Hamzaoui has been more than an advisor to me during my stay at the University of Konstanz. I dearly appreciate his kindness and generosity.

Additionally, I wish to thank Dr. Magnus Nilsson of the University of Kalmar for his guidance and suggestions during my thesis work.

The have been many individuals who have aided in my intellectual development here at the University of Konstanz. I specially wish to thank Shakeel Ahmad for his help and timely advices. I also wish to thank all of my peers of the Multimedia Signal Processing Group at the University of Konstanz, namely Martin Röder, Tilo Ochotta, Mauro Ruggeri, Vladimir Bondarenko and Martin Preiser for their friendship and guidance. I also wish to thank Anna Dowden-Williams for her close friendship and encouragement.

Special thanks to Professor Dr. Dietmar Saupe, who allowed me to conduct my thesis work at the University of Konstanz.

Last, but my no means least, I want to thank my family and close friends Iulia Guriuc, Satyajit Saste and Sudesh Kulkarni for their love and support they continue to provide at all times. I will never be able to adequately thank my parents Charulata and Bharatkumar Borlikar, for their guidance, strength, and the love they have given me all the years of my life. Everything I do or become is a testament to the wonderful upbringing I received from them.

# Contents

# Chapter 1

# INTRODUCTION

## Summary

In this chapter we provide the motivation and brief introduction with a background about video streaming and scalable video compression. We conclude this chapter with the problem statement and the outline to this thesis.

## 1.1 Motivation and background

Over the past fifty years, videos, movies, and television have become an increasingly important part of our culture, used in entertainment, advertising, education, as an art form, in the news, etc. Naturally, the Internet and intranet communities, in which innovative visual presentations are an essential element for success, would like to take advantage of video technology and make it work for them, too. More and more Web sites are beginning to do exactly that, and are using audio and video streaming technology to do it. Streaming media is relatively easy for companies to add to their Web sites, and brings video and audio content to a potentially wider audience much more cost-effectively than other broadcast methods. Consumers can tune in at any time with readily available software, such as the RealPlayer and a host of other programs.

Streaming is a method of digitizing and delivering an audio or video file from a remote server to your PC. Just as radio and television broadcasts are not downloaded and stored on radios and TVs, a streaming file is not actually stored on the computer used to view it. Instead, the video player software on the user's computer continually requests video data from a server, creating a

buffer so that the user does not have to wait for the entire file to download before viewing.

When low bandwidth limits the client's ability to receive data, a scalable technology will ensure an unbroken audio stream by scaling the amount of video data transmitted. The simplest technique is stream thinning, in which the server doesn't transmit every video frame to the client and sends less information per video frame, losing image quality while preserving the frame delivery rate. Most streaming-video technologies use a server to ensure efficient video distribution to clients. Servers transmit video through a variety of network protocols including UDP, TCP and HTTP.

Anyone already using video in some context should seriously consider using streaming media. If they're already using regular downloadable media, streaming media should definitely be in their plans. No one likes to wait to download audio or video. A middle of the road solution is to use both streaming and downloadable media.

This project report is an insight into the world of video streaming. It gives an introduction to real time video streaming and also deals with the necessary logic behind its implementation. Finally we express the necessity of video streaming in a bigger picture.

## 1.2 Problem Definition and Thesis Outline

With the explosive growth of video applications over the Internet, many approaches have been proposed to stream video effectively over packet-switched networks. A number of these use techniques from source and channel coding, or implement transfer protocols, or modify system architectures in order to deal with delay, loss and time varying nature of the Internet.

The goal of the project is to design a system that can feed a live video from the server upon a request from a client over the Internet. The feed is initially captured from a camera. Scalable video has become a very important topic of research as many applications regarding multimedia transmission are evolving over the Internet. The video is compressed according to the H.263 video compression standard. The video that is transferred is error protected with Digital Fountain Codes (Luby Transform Codes). These codes were chosen due to their simplicity in designing the encoding and decoding algorithm. The most important advantage behind using these codes is that an encoding symbol can be generated on the fly. Unlike the Raptor Codes where the source symbols can be recovered when the same length of symbols has reached the receiver Luby Transform Codes require a small amount of overhead (5% - 10%).

The thesis is structured as follows

This chapter presented a brief introduction to the problem. The main purpose of this chapter is to give a basic idea about the thesis and how the system would be designed. Chapter 2 gives a detailed explanation of the video compression, how it is implemented and specially the H.263 video compression standard. In chapter 3 we see about Error Protection Codes and why we chose Digital Fountain Codes for the error protection of the compressed bitstream and how the algorithms are designed to make these codes effective in real time video transmission with some experimental results. In chapter 4 we discuss about streaming protocols and discuss is detail about User Datagram Protocol (UDP) and the reason behind using this protocol for our transmission and the advantages of using this protocol instead of the Transmission Control Protocol. We also discuss how Real Time Streaming Protocol and see how this protocol could be run over UDP. In chapter 5 the whole system design is discussed with experimental results. Chapter 6 concludes our thesis and discusses the future scope.

# Chapter 2

# VIDEO CODING

**Summary**

In this chapter we first see the need for scalable video compression and then discuss in detail the H.263 video compression standard considering video frame structure, coding tools and see how motion estimation and compensation is done. We also have a look how the entropy coding is done in this compression standard. Further we study the different modes available for transmission in the H.263 standard. We finally discuss the implementation issues of this standard in real time video transmission.

## 2.1 Motivation for Digital Video Compression

Digital technologies are omnipresent in our society. Many analog appliances and technologies are being converted to a binary world of ones and zeros. In less than a decade television as we know today will no longer exist. Publicly streamed digital television is the near future. Upon its arrival television viewers will have a larger number of channels, each with increased resolution and picture quality. Digital video compression makes such improvements possible.

In addition to television, it is now quite evident that the age of communication has arrived; people have raised their expectations and now expect phones providing full visual communication at a good resolution as opposed to the conventional telephone. However, bandwidth limitations imposed by our existing communication infrastructure makes high quality video transmission quite difficult. These limitations give video compression a dominant and important role in enabling digital video communication. Without video

compression, land-line visual communication would not merely be difficult but impossible.

To further demonstrate the need of video compression, consider the case of a quarter common intermediate format (QCIF) sequence (176 * 144) with 30 frames per second. Since QCIF defines the image representation to use 4:2:0 format, there are effectively 12 bits per pixel. With this 4:2:0 format both chrominance values are downsampled by a factor of two in each dimension. Thus a single 8 bit chrominance corresponds to four pixels. The 12 bit average per pixel is determined by adding 8 bits per pixel for luminance and 8 bits per 4 pixels for both values of chrominance. Using these specifications the uncompressed bitrate required for transmission of the uncompressed sequence is:

176 x 144 x 30 x 12 = 9.1 Mb/sec

This bitrate is very high for the transmission over our existing networks. Thus video compression is a critical criterion in today's digital world.

The most important application, which relies on digital compression technology, is the Internet. The World Wide Web (WWW) has connected users across the globe and enabled them to convey information in many forms: text, sound, images and video. Image and video compression have enabled the WWW to the serve the information needs of our increasingly multimedia focused world.

## 2.2 Definition of Video Compression

The main objective of video compression or coding is to decrease the file size required to store or to bitrate required to transmit video sequences. Video compression algorithms can be classified into two types: lossy and lossless. Lossless compression reduces the redundancy found in the representation of video sequences without introducing any degradation. While such schemes

allow exact representation of the video sequence, they do not allow enough compression to transmit the sequences without incredible delays. Thus we are focused to use lossy video compression schemes where the compressed representation does not exactly match the original video sequence. The obvious goal of lossy video compression algorithms is to provide the highest quality visual representation of the original video for a given bitrate. Ultimately video compression algorithms hope to achieve visually lossless compression in which degradation introduced by the lossy compression is not noticeable to the viewers. Thus, for lossy video compression, we reduce redundancy of video sequences and permit a less than perfect representation of the sequence in areas where the degradation is less likely to be noticed.

## 2.3 Definition of Scalable Video Compression

Scalable video compression involves incorporating another goal or objective into lossy video compression. Namely, scalable video compression involves embedding video sequences of decreased quality or resolution within an overall compressed bitstream. Naturally subsets of increased size (larger portion of the overall compressed bitstream) correspond to sequences of increased quality or higher resolution. Scalable video compression allows a single compressed bitstream to meet the needs of various users with different constraints. Therefore, scalable coding allows storage or transmission of highest quality video sequence that resources will allow.

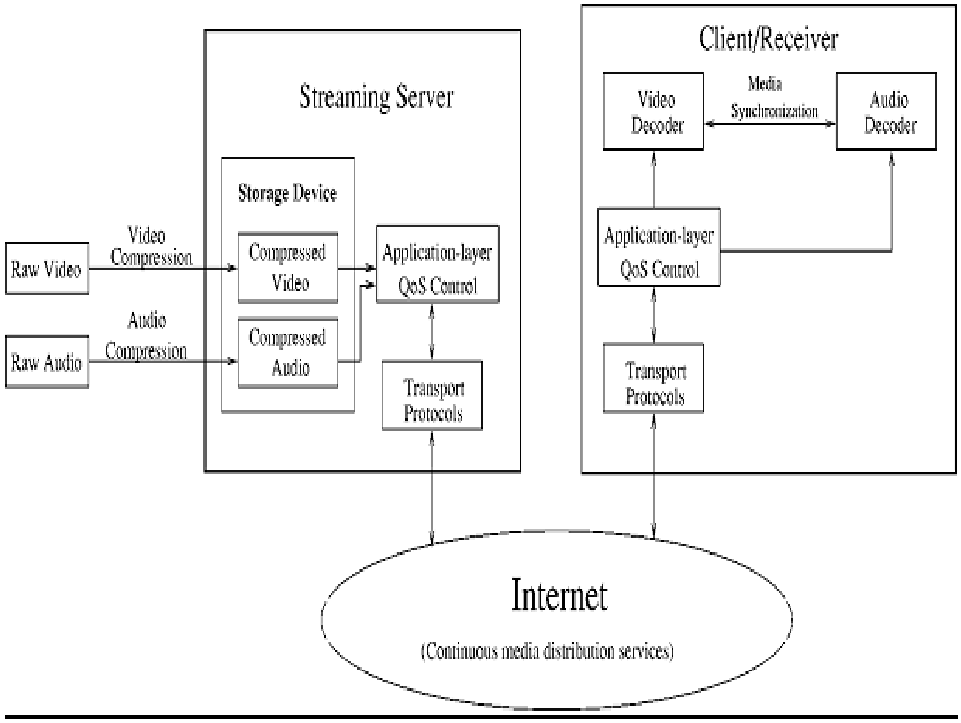
## 2.4 Initial Work in Video Compression

Initially digital images were much more common place than digital video sequences and therefore initial steps for video compression involved adaptations of well accepted image compression techniques. One of the simplest video compression algorithms is called motion JPEG and it involves independently coding the frames of a video sequence using the JPEG

algorithm. While the computational simplicity of the approach provided realtime capabilities, it fails to attain the necessary compression efficiencies due to the fact that frames are coded independently; thus, the temporal redundancy inherent to adjacent frames is not utilized to improve the compression. The motion JPEG concept of independently coding each frame can be used with any image compression technique, but none of the approaches will be competitive with motion compensated video coding.

Earlier approaches that constituted to International standards include H.261, MPEG-1 and MPEG-2. All these three standards utilize a block-based motion compensated compression using DCT. MPEG-1 and MPEG-2 target higher bitrate compression (1Mb/sec and above). H.261 is a low bitrate compression standard aimed at providing compression at multiples of 64 kb/sec. Since H.261 bears much resemblance H.263 we use this standard, which offers better compression at low bitrates.

## 2.5 Block Diagram and Description




This project is a prototype of Real Time Video Streaming with H.263 Encoding and Decoding (Video CODEC) and protecting the Stream with Digital Fountain Codes against packet loss.

For the realization of the above-mentioned prototype, the Web Camera is interfaced with a computer, the protocol of which is implemented in the firmware.

The high-speed transmission is achieved using the FFMPEG software which is an open source. The specifications given in the configuration file depict the specifications of the video characteristic as a whole. In-order to simulate video streaming, first the FFSERVER is started in which the encoding details are

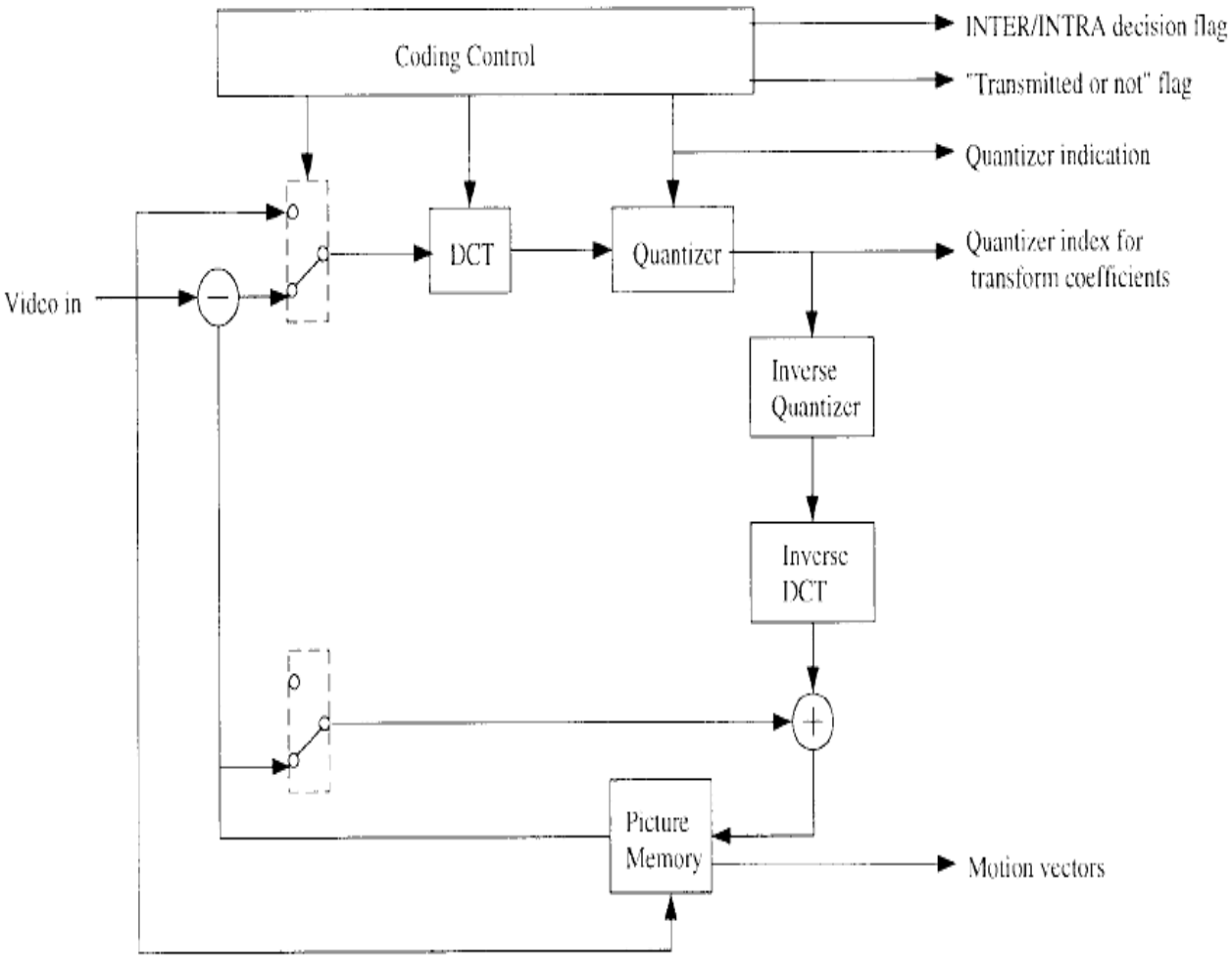
initialized. The FFSERVER.CONF, which is the configuration file in which the settings of the video content are set is the standard for the video. We can now start the encoding of the video using our encoding command. Once the encoding has started which is a H263 encoded file, we need to protect this file against loss over the transmission channel. So now this is where the Digital Fountain Codes come into play. We protect our stream with these Rateless Erasure Codes against transmission loss.

The receiver side consists of a decoder for the Digital Fountain Codes and a decoder for the H263 encoded video. Once all the decoding is done in an appropriate way we can watch the video on our screen using the Real Player and just asking for the location of the video from the server. The video can then be watched at real time.

## 2.6 H.263 Video Compression Standard

A number of video coding standards (H261, H263 and the latest H264/AVC) exist, each of which is designed for a particular type of application: for example, MPEG2 for digital television and H.261 for ISDN video conferencing. These standards address a wide range of applications having different requirements in terms of bit rate, picture quality, complexity, error resilience, and delay. H.263 is aimed particularly at video coding for low bit rates (typically 20-30 kbps) and above. The H.263 standard specifies the requirements for a video encoder and decoder. It does not describe the encoder or decoder itself: instead, it specifies the format and content of the encoded (compressed) stream.

## 2.7 Baseline H263 Video coding




**Video Encoder block diagram for H263**

Motion-compensated prediction first reduces temporal redundancies. Discrete cosine transform (DCT)-based algorithms are then used for encoding the motion-compensated prediction difference frames. The quantized DCT coefficients, motion vectors, and side information are entropy coded using variable length codes (VLC's).

### 2.7.1 Video frame Structure

H.263 supports five standardized picture formats: sub-QCIF, QCIF, CIF, 4CIF, and 16CIF. The luminance components of the picture is sampled at these resolutions, while the chrominance components, and, are down sampled by two in both the horizontal and vertical directions.

The picture shown on the next page is for QCIF resolution. Each picture in the input video sequence is divided into macroblocks, consisting of four luminance blocks of 8 pixels * 8 lines followed by one **Cb** block and one **Cr** block, each consisting of 8 pixels * 8 lines. A group of blocks (GOB) is defined as an integer number of macroblock rows, a number that is dependent on picture resolution.

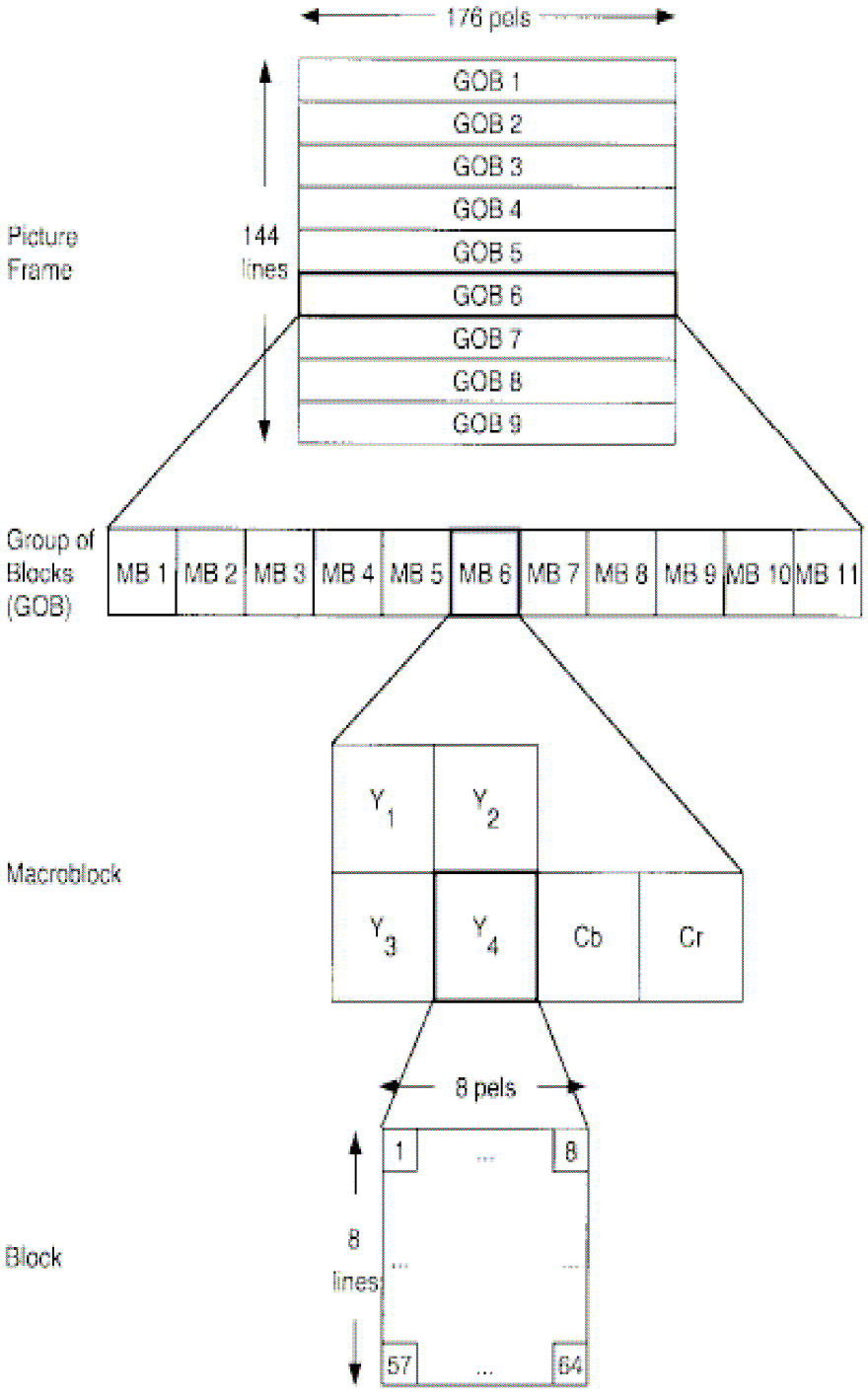
### 2.7.2 Video Coding Tools

H.263 supports interpicture prediction that is based on motion estimation and compensation. The coding mode where temporal prediction is used is called an inter mode. In this mode, only the prediction error frames—the difference between original frames and motion-compensated predicted frames—need be encoded. If temporal prediction is

not employed, the corresponding coding mode is called an intra mode.

#### 2.7.2.1 Motion estimation and compensation

Motion-compensated prediction assumes that the pixels within the current picture can be modeled as a translation of those within a previous picture. In baseline H.263, each macroblock is predicted from the previous frame. This implies an assumption that each pixel within the macroblock undergoes the same amount of translational motion.




**QCIF Resolution**

This motion information is represented by two dimensional displacement vectors or motion vectors. Due to the block-based picture representation, many

motion estimation algorithms employ block-matching techniques, where the motion vector is obtained by minimizing a cost function measuring the mismatch between a candidate macroblock and the current macroblock. Although several cost measures have been introduced, the most widely used one is the sum-of-absolute-differences (SAD) defined by

$$\mathbf{SAD} = \sum_{k=1}^{16} \sum_{l=1}^{16} \boldsymbol{B}_{i,j}(k,l) - \boldsymbol{B}_{i-u,j-v}(k,l)$$

Where $\boldsymbol{B}_{i,j}(k,l)$ represents the $(k,l)^{th}$ pixel of a 16*16 macroblock from the current picture at the spatial location (i,j), and $\boldsymbol{B}_{i-u,\,j-v}(k,l)$ represents the $(k,l)^{th}$ pixel of the candidate macroblock from a reference picture at the spatial location $(i,j)$ displaced by the vector $(u,v)$. To find the macroblock producing the minimum mismatch error, we need to calculate the SAD at several locations within a search window. The simplest, but the most compute-intensive search method, known as the full search or exhaustive search method, evaluates the **SAD** at every possible pixel location in the search area. To lower the computational complexity, several algorithms that restrict the search to a few points have been proposed. In baseline H.263, one motion vector per macroblock is allowed for motion compensation. Both horizontal and vertical components of the motion vectors may be of half pixel accuracy, but their values may lie only in the [ 16, 15.5] range, limiting the search window used in motion estimation. A positive value of the horizontal or vertical component of the motion vector represents a macroblock spatially to the right or below the macroblock being predicted, respectively.

**Transform**: The purpose of the 8 * 8 DCT specified by H.263 is to decorrelate the 8 * 8 blocks of original pixels or motion-compensated

difference pixels, and to compact their energy into as few coefficients as possible. Besides its relatively high decorrelation and energy compaction capabilities, the 8 * 8 DCT is simple, efficient, and amenable to software and hardware implementations. The most common algorithm for implementing the 8 * 8 DCT is that which consists of eight-point DCT transformation of the rows and the columns, respectively. The 8 * 8 DCT is defined by

$$\mathbf{C_{m,n}} = \alpha(m)\beta(n) \sum_{i=1}^{8} \sum_{j=1}^{8} \mathrm{B_{i,j}} \cos\left(\frac{\pi(2i+1)m}{16}\right)$$

$$\bullet \cos\left(\frac{\pi(2j+1)n}{16}\right), \quad 0 \le m, \mathrm{n} \le 7$$

Where $\alpha(0) = \beta(0) = \sqrt{\frac{1}{8}}$ and $\alpha(m) = \beta(n) = \sqrt{\frac{1}{4}}$

For $1 \le m, n \le 7$

Here, $\mathbf{B_{i,j}}$ denotes the $(i,j)^{th}$ pixel of the 8 * 8 original block, and $\mathbf{C_{m,n}}$ denotes the coefficients of the 8 * 8 DCT transformed block. The original 8 * 8 block of pixels can be recovered using an 8 * 8 inverse DCT (IDCT) given by

$$\mathrm{B_{i,j}} = \sum_{m=1}^{8} \sum_{n=1}^{8} \mathrm{C_{m,n}}\ \alpha(m) \cos\left(\frac{\pi(2m+1)i}{16}\right) \beta(n)$$

$$\bullet \cos\left(\frac{\pi(2n+1)j}{16}\right), \quad 0 \le i, \mathrm{j} \le 7$$

Although exact reconstruction can be theoretically achieved, it is often not possible using finite-precision arithmetic. While forward DCT errors can be

tolerated, inverse DCT errors must meet the H.263 standard if compliance is to be achieved.

### 2.7.2.2 Quantization

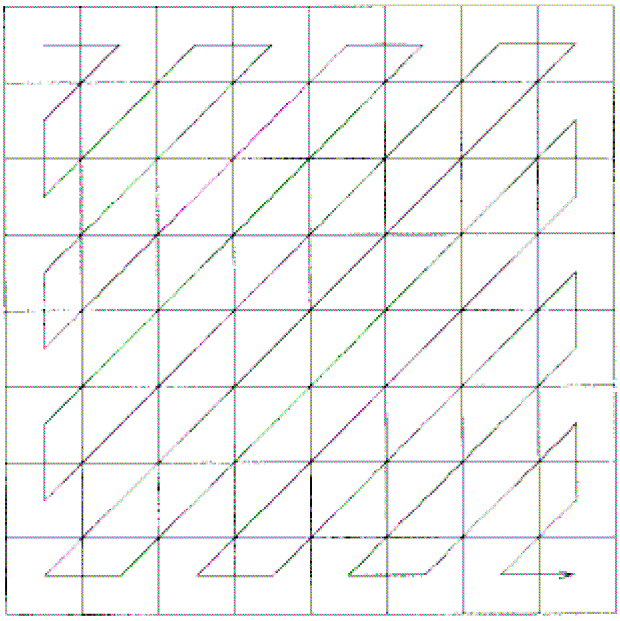
The human viewer is more sensitive to reconstruction errors related to low spatial frequencies than those related to high frequencies. Slow linear changes in intensity or color (low-frequency information) are important to the eye. Quick, high frequency changes can often not be seen, and may be discarded. For every element position in the DCT output matrix, a corresponding quantization value is computed using the equation

$$\mathbf{C^q_{m,n} = C_{m,n} / Q_{m,n}} \quad 0 \le m, n \le 7$$

Where $\mathbf{C}_{m,n}$ is the $(m,n)^{th}$ DCT coefficient and $\mathbf{Q}_{m,n}$ is the $(m,n)^{th}$ quantization value. The resulting real numbers are then rounded to their nearest integer values. The net effect is usually a reduced variance between quantized coefficients as compared to the variance between the original DCT coefficients, as well as a reduction of the number of nonzero coefficients. In H.263, quantization is performed using the same step size within a macroblock (i.e., using a uniform quantization matrix). Even quantization levels in the range from 2 to 62 are allowed, except for the first coefficient (DC coefficient) of an intra block, which is uniformly quantized using a step size of eight. The quantizers consist of equally spaced reconstruction levels with a dead zone centered at zero. After the quantization process, the reconstructed picture is stored so that it can be later used for prediction of the future picture.

### 2.7.2.3 Entropy Coding

Entropy coding is performed by means of variable length codes (VLC's). Motion vectors are first predicted by setting their component's values to median values of those of neighboring motion vectors already transmitted: the motion vectors of the macroblocks to the left, above, and above right of the current macroblock. The difference motion vectors are then VLC coded. Prior to entropy coding, the quantized DCT coefficients are arranged into a one-dimensional array by scanning them in zigzag order. This rearrangement places the DC coefficient first in the array, and the remaining AC coefficients are ordered from low to high frequency. This scan pattern is shown as below.



The rearranged array is coded using a three-dimensional run-length VLC table, representing the triple (LAST, RUN, LEVEL). The symbol RUN is defined as the distance between two nonzero coefficients in the array. The

symbol LEVEL is the nonzero value immediately following a sequence of zeros. The symbol LAST replaces the H.261 endof block flag, where
"LAST = 1" means that the current code corresponds to the last coefficient in the coded block. This coding method produces a compact representation of the 8 * 8 DCT coefficients, as a large number of the coefficients are normally quantized to zero and the reordering results (ideally) in the grouping of long runs of consecutive zero values. Other information such as prediction types and quantizer indication is also entropy coded by means of VLC's.

#### 2.7.2.4 Coding Control

The performance of the motion estimation process, usually measured in terms of the associated SAD values, can be used to select the coding mode (intra or inter). If a macroblock does not change significantly with respect to the reference picture, an encoder can also choose not to encode it, and the decoder will simply repeat the macroblock located at the subject macroblock's spatial location in the reference picture.

## 2.8 Optional Modes

In addition to the core encoding and decoding algorithms described above, H.263 includes four negotiable advanced coding modes: unrestricted motion vectors, advanced prediction, *PB* frames, and syntax-based arithmetic coding. The first two modes are used to improve inter picture prediction. The *PB*-frames mode improves temporal resolution with little bit
rate increase. When the syntax-based arithmetic-coding mode is enabled, arithmetic coding replaces the default VLC coding. These optional modes allow developers to trade off between compression performance and complexity. We now discuss in brief about these four modes.

### 2.8.1 Unrestricted Motion Vector Mode

In baseline H.263, motion vectors can only reference pixels that are within the picture area. Because of this, macroblocks at the border of a picture may not be well predicted. When the unrestricted motion vector mode is used, motion vectors can take on values in the range [31.5, 31.5] instead of [16, 15.5], and are allowed to point outside the picture boundaries. The longer motion vectors improve coding efficiency for larger picture formats, i.e., 4CIF or 16CIF. Moreover, by allowing motion vectors to point outside the picture, a significant gain is achieved if there is movement along picture edges. This is especially useful in the case of camera movement or background movement.

### 2.8.2 Syntax-Based Arithmetic Coding Mode

Baseline H.263 employs variable-length coding as a means of entropy coding. In this mode, syntax-based arithmetic coding is used. Since VLC and arithmetic coding are both lossless coding schemes, the resulting picture quality is not affected, yet the bit rate can be reduced by approximately 5% due to the more efficient arithmetic codes. It is worth noting that use of this annex is not widespread.

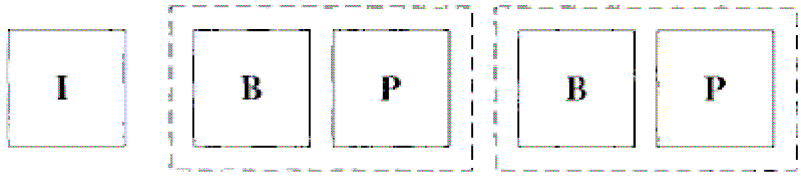
### 2.8.3 Advanced Prediction Mode

This mode allows for the use of four motion vectors per macroblock, one for each of the four 8 * 8 luminance blocks. Furthermore, overlapped block motion compensation is used for the luminance macroblocks, and motion vectors are allowed to point outside the picture as in the unrestricted motion vector mode. Use of this mode improves inter picture prediction, and yields a significant improvement in subjective picture quality for the same bit rate by reducing blocking artifacts.

### 2.8.4 *PB* Frames Mode

In this mode, the frame structure consists of a *P* picture and a *B* picture,

Example like shown below: -




The quantized DCT coefficients of the *B* and *P* pictures are interleaved at the macroblock layer such that a *P*-picture macroblock is immediately followed by a *B*-picture macroblock. Therefore, the maximum number of blocks transmitted at the macroblock layer is 12 rather than 6. The *P* picture is forward predicted from the previously decoded *P* picture. The *B* picture is bidirectionally predicted from the previously decoded *P* picture and the *P* picture currently being decoded. The forward and backward motion vectors for a *B* macroblock are calculated by scaling the motion vector from the current *P*-picture macroblock using the temporal resolution of the *P* and *B* pictures with respect to the previous *P* picture. If this motion vector does not yield a good prediction, a delta vector can enhance it. The delta vector is obtained by performing motion estimation, within a small search window,
in approximation with the calculated motion vectors. When decoding a *PB*-frame macroblock, the *P* macroblock is reconstructed first, followed by the *B* macroblock since the information from the *P* macroblock is needed for *B* macroblock prediction. When using the *PB*-frames mode, the picture rate can be doubled without a significant increase in bit rate.

## 2.9 Implementation Issues for Real Time Video Transmission

### 2.9.1 Bit Rate Control

Practical communications channels have a limit to the number of bits that they can transmit per second. In many cases the bit rate is fixed. The basic H.263 encoder generates a variable number of bits for each encoded frame. If the motion estimation/compensation process works well then there will be few remaining non-zero coefficients to encode. However, if the motion estimation does not work well (for example when the video scene contains complex motion), there will be many non-zero coefficients to encode and so the number of bits will increase.

In order to "map" this varying bit rate to (say) a CBR channel, the encoder must carry out **rate control**. The encoder measures the output bit rate of the encoder. If it is too high, it increases the compression by increasing the quantizer scale factor: this leads to more compression (and a lower bit rate) but also gives poorer image quality at the decoder. If the bit rate drops, the encoder reduces the compression by decreasing the quantizer scale factor, leading to a higher bit rate and a better image quality at the decoder.

### 2.9.2 Synchronization

The encoder and decoder must stay in synchronization, particularly if the video signal has accompanying audio. The H.263 bitstream contains a number of "headers" or markers: these are special codes that indicate to a decoder the position of the current data within a frame and the "time code" of the current frame. If the decoder loses synchronization then it can "scan" forward for the next marker in order to resynchronize and resume decoding. It should be noted

that even a brief loss of synchronization can cause severe disruption in the quality of the decoded image and so special care must be taken when designing a video coding system to operate in a "noisy" transmission environment.

### 2.9.3 Audio and Multiplexing

The H.263 standard describes only video coding. In many practical applications, audio data must also be compressed, transmitted and synchronized with the video signal. Synchronization, multiplexing and protocol issues are covered by "umbrella" standards such as H.320 (ISDN-based videoconferencing), H.324 (POTS-based videotelephony) and H.323 (LAN or IP-based videoconferencing). H.263 (or its predecessor, H.261) provide the video coding part of these standards groups. A range of standards including G.723.1 supports audio coding. Other, related standards cover functions such as multiplexing (e.g. H.223) and signaling (e.g. H.245).

## 2.10 Applications of H.263 Video Codec

Videoconferencing and videotelephony have a wide range of applications including:

- Desktop and room-based conferencing.
- Video over the internet and telephone lines.
- Surveillance and monitoring.

In each case video information (and perhaps audio as well) is transmitted over telecommunications links, including networks, telephone lines. Video has a high "bandwidth" (i.e. many bytes of information per second) and so these applications require **video compression** or **video coding** technology to reduce the bandwidth before transmission. This is exactly the reason why H263 has been developed.

# Chapter 3

# Error Protection Codes


## Summary

In this chapter we have a look at the need for error protection and discuss Digital Fountain Codes with detailed explanation of Luby Transform Codes. We see in detail the encoder and decoder algorithms work with an example of how decoding is done using these codes. The degree distribution that we are using is also discussed in detail with experimental results and finally we conclude this chapter by the having a look at the advantages of using these sparse graph codes for real time video transmission.


## 3.1 Introduction to Digital Fountain Codes

Digital fountain codes are sparse-graph codes for channels with erasures.

Files sent over the Internet are chopped into packets, and each packet is either received without error or not received. Hence channels with erasure are of great importance.

One of the policies adopted for such channels is the policy that is used in Transmission Control Protocol, which is using acknowledge signals from receiver to transmitter for sending again packets that are lost in transmission.

This method adopted has the advantage that it works no matter how lossy the channel may be. This is very old approach and universal belief that this will not work properly in the future ahead as it is very time consuming for real time networks. This is more evident in broadcast networks where retransmission protocols will be so many if all the users do not receive all the packets properly that the whole information being broadcasted would have to be retransmitted again.

This is the reason why we need correcting codes, which require very less information from the transmitter. We need randomized constructions of linear-time encodable and decodable codes that can transmit over lossy channels at rates extremely close to capacity. The encoding and decoding algorithms for these codes must have fast implementation. Such codes will be extremely useful for real-time audio and video transmission over the Internet, where lossy channels are common and fast decoding is a premium requirement.

Reed Solomon Codes are one such type of codes. They have one disadvantage that they are efficient only for a small size of (n, k) where n is the length and k is the dimension of the code. One more thing that Reed-Solomon codes have to follow is that the erasure probability has to be predicted before hand. The code rate must also be determined. These requirements make these codes not that efficient. These shortages would prove not to be efficient for Real Time applications. Digital Fountain Codes are codes that will overcome this shortage. Michael Luby produced them in 1998.

## 3.2 Luby Transform Codes

LT codes are one of the first class of erasure codes that we call universal erasure codes. The symbol length for the codes can be arbitrary, from one-bit binary symbols to general *L*-bit symbols. We analyze the run time of the encoder and decoder in terms of symbol operations, where a symbol operation is either an exclusive-or of one symbol into another or a copy of one symbol to another. If the original data consists of *K* input symbols then each encoding symbol can be generated, independently of all other encoding symbols.

LT codes are rateless, i.e., the number of encoding symbols that can be generated from the data is potentially limitless. Furthermore, encoding symbols can be generated on the fly, as few or as many as needed. The decoder can recover an exact copy of the data from any set of the generated encoding symbols that in aggregate are only slightly longer in length than the data. Thus, no matter which channel it is, encoding symbols can be generated

as needed and sent over the erasure channel until a sufficient number have arrived at the decoder in order to recover the data. Since the decoder can recover the data from nearly the minimal number of encoding symbols possible, this implies that LT codes are near optimal with respect to any erasure channel. **Luby Transform Codes (LT Codes)** are a part of **Digital Fountain Codes.** Let us have a look in detail how do these LT codes work.

The idea of these codes is that the encoder is a fountain that produces endless supply of encoded symbols. E.g. the original source file has a size of kl bits, and each symbol contains l-encoded bits. Anyone who wants to receive the compressed file collects packets, which are slightly more than the original size of the file. Now the file can be decoded. Basically the original file can be decoded with 5% more number of symbols from the original data. The decoding and encoding complexities are also quite small compared to the other codes. They are very efficient as the file size k grows. The overhead k´ - k is of the order $\sqrt{k}$ (ln (k/$\delta$ ))$^2$.

## 3.3 The Encoder

### Algorithm:

An encoded stream $t_1,t_2,t_3\ldots..t_n$ can be produced from source symbols $s_1,s_2,s_3\ldots.s_k$ as follows.

- The degree $d_n$ of each source symbol can be chosen randomly from the Degree Distribution $\rho$ (d).
- Choose $d_n$ distinct input source (source packets) uniformly at random, and set $t_n$ equal to the bitwise sum (modulo 2) of those symbols.

Basically what we get from this is a graph showing the connections of encoded symbols with the source symbols. As we have mentioned in the introduction about LT codes and that they are Sparse Graph codes, let us

discuss a little about Sparse Graphs. A graph with relatively few edges is *sparse.*

A *sparse graph* is a graph $\mathbf{G = (\nu, \varepsilon)}$ in which $|\varepsilon| = (|\nu|)$.

For example, consider a graph $\mathbf{G = (\nu, \varepsilon)}$ with *n* nodes. Suppose that the out-degree of each vertex in *G* is some fixed constant *k*. Graph *G* is a *sparse graph* because $|\varepsilon| = k|\nu| = (|\nu|)$.

## 3.4 The Decoder

Decoding sparse graph codes is relatively easy. The decoder basically needs to know which encoded symbols are connected to which source symbols and how many source symbols. This information needs to be sent to the decoder. The sender can give a key to the decoder, which tells him the appropriate details of the connections of the graph. The key should be of a small size to avoid any delay in decoding. Other way is both the encoder and decoder could have synchronized clocks.

Now we can evaluate the decoder such that it would have to recover $s_n$ symbols from $\mathbf{t_n = G * s_k}$ where G is the connection matrix showing all the connections between encoded symbols and source symbols.

## Algorithm:

- We first have to find an encoded symbol $t_n$ which is connected to only source symbol.

  i.e. with degree one. This is the main criterion of this algorithm. If there is no such node then the decoder cannot recover the source data.

  (a) Set $s_k = t_n$

  (b) Add $s_k$ to all checks $t_{n'}$ that are connected to $s_k$:

  $$t_{n'} := t_{n'} + s_k \text{ for all } n' \text{ such that } G_{n',k} = 1.$$

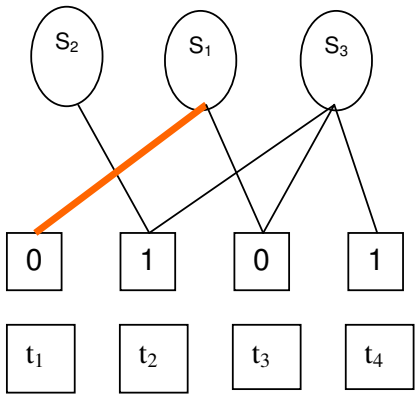
  (c) Remove all the edges connected to the source symbol $s_k$.

- Repeat the above steps until all the source symbols are determined.

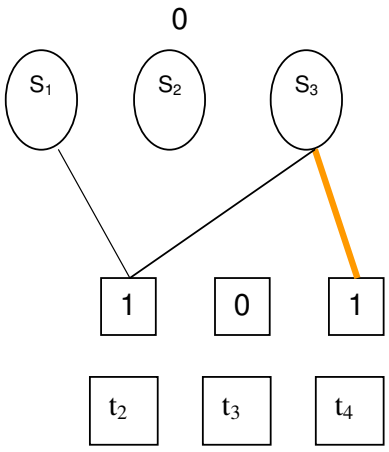
### 3.4.1 Example of LT decoding

The decoding can be illustrated as shown below.

Suppose we have this condition.




Here the first encoded symbol $t_1$ has degree one. Bit assigned to source symbol $s_2$ is 0. Now we see for links of $s_2$.

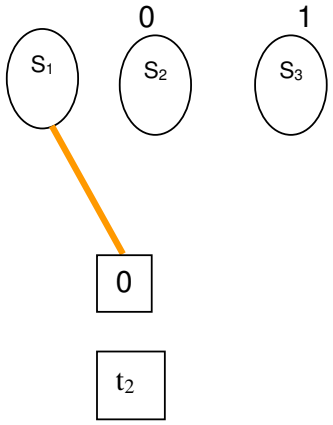



Now we do modulo2 addition of 0 the bit assigned to $s_2$ and the ends of the links $t_3$.

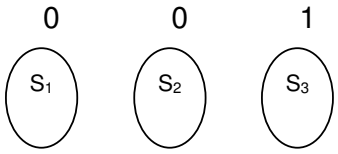
0 XOR 0 = 0

Hence $t_3$ remains 0.

Now the decoding algorithm is run again. The next bit is $t_2$ . The degree of this symbol is not 1. So the decoding algorithm does not run. It waits for the next symbol to arrive. When $t_3$ is received since the degree is zero, the algorithm does not run. It waits for the next symbol to arrive. When symbol $t_4$ is received, as it has degree 1 the decoding algorithm runs and assigns 1 to source symbol $s_3$. Now the result of modulo 2 addition of the bit assigned (1) to $s_3$ and the ends of the links $t_2$.




As we see the symbol $t_2$ has degree one, hence 0 is assigned to source symbol $s_1$. We have now recovered the all the source symbols.



## 3.5 Degree Distribution:

The probability distribution $\rho$ (d) is a critical part of design.

Occasional encoded symbols must have high degree (similar like k) just as a safety or precaution that there are not some source symbols, which are connected to no channel symbols.

Many symbols should have low degree. This should be a very critical condition so that the decoding can start. This will also ensure that total number of additions while encoding and decoding is small.

In an ideal case the graph received at the receiver should be in such a manner that at every iteration that we carry out we should have only a single degree one symbol. Now when this symbol is under scrutinisation the degrees of the graph are reduced in such a way that a single new degree one symbol would come into consideration.

## Two types of Degree Distribution:

1. Ideal Soliton Distribution
2. Robust Soliton Distribution.

### 3.5.1 Ideal Soliton Distribution:

$\rho$ (1) = 1/k

$\rho$ (d) = 1/d (d-1) for d= 2,3,…, k.

The expected degree distribution is roughly ln k.

## Disadvantage:

1. Fluctuations around the expected behavior results sometimes in no degree-one check symbols during decoding.
2. Few source symbols will receive no connections at all.

### 3.5.2 Robust Soliton Distribution:

Two extra parameters c and $\delta$ ensure that expected number of degree-one checks is about

$$S=c \ln (k/\delta) \sqrt{k}$$

$\delta$ is bound on probability that decoding fails to run to completion.

c is constant of order 1,according to Luby's theorem.

This could also have a value slightly smaller than one. This value yields better result than c = 1.

Now we define a positive function: -

$$t(d) = \begin{cases} S/kd & \text{for } d=1,2\ldots(k/S)-1 \\ (S/k) \ln (S/d) & \text{for } d=k/S \\ 0 & \text{for } d>k/S \end{cases}$$

Now if we add the ideal soliton distribution $\rho$ to $\tau$ and normalize to obtain the robust soliton distribution, $\mu$:

$$\mu(d) = [\rho(d)+\tau(d)]/Z$$

Where $Z=\sum_{d} \rho(d)+\tau(d)$.

The number of encoded nodes required at the receiving end to ensure that the decoding can run to completion, with probability at least

1 - $\delta$, is k´ = kZ.

### Experimental Results:

Result obtained in the expeiment: -

K=10,000,c=0.2, $\delta$ = 0.05 and S comes out to be 243.67,K/S=41.1 and

$Z \approx 1.29$.

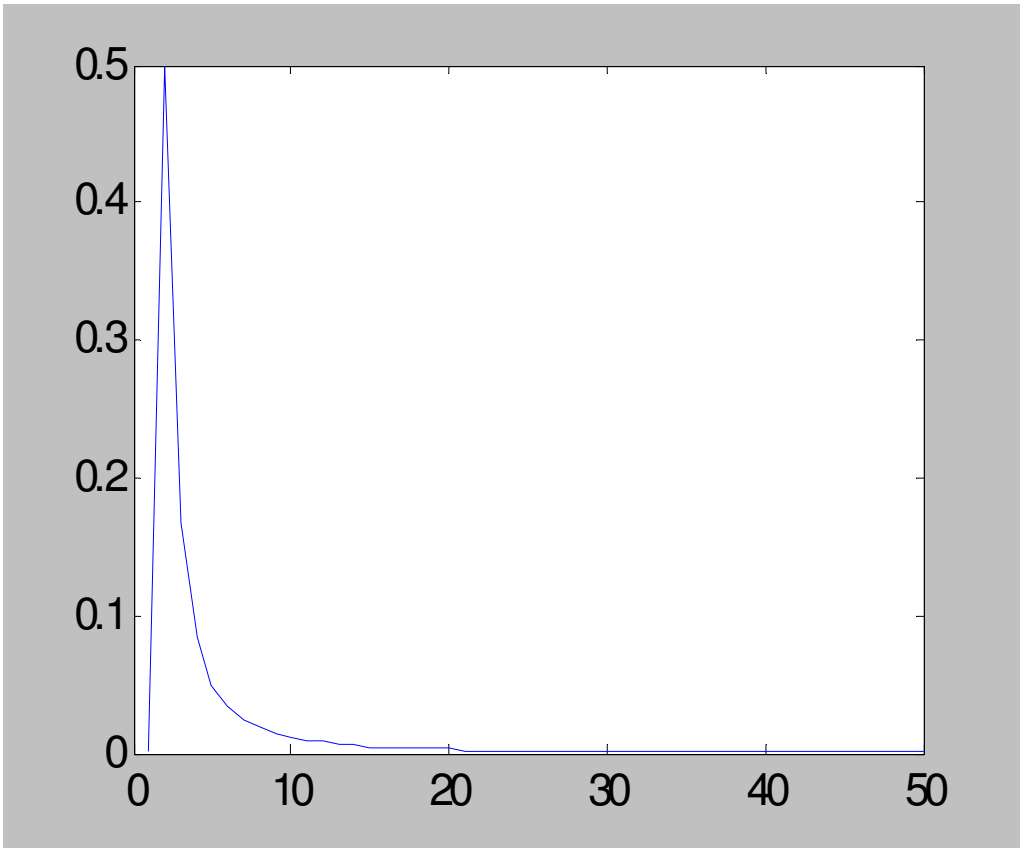



Figure for rho

## 3.6 Advantages of Digital Fountain Codes

- Small encoding and decoding complexities.
- Encoding symbols can be generated on the fly, as few or as many as needed.
- Decoder can recover an exact copy of the data from any set of the generated encoding symbols with only slightly longer in length than the data.

# Chapter 4

# Transmission Protocols


## Summary

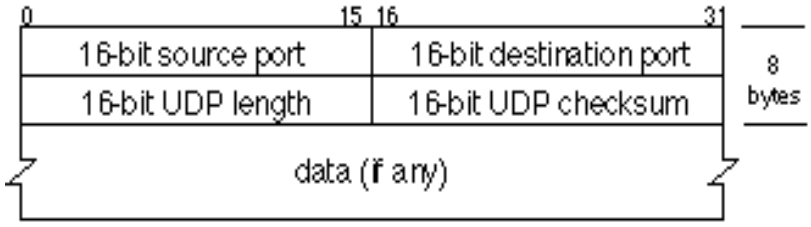
In this chapter the transmission protocols used in our system are discussed. First we discuss in detail the User Datagram Protocol considering its header structure, how checksum in done in UDP and see how UDP is a connectionless protocol and compare it with TCP and see the advantages of using UDP in real time video transmission. Later we also have a look at some other multimedia streaming protocols and discuss in detail the Real Time Streaming Protocol (RTSP). We see how the prefix caching in done in RTSP, with the basic RTSP operation. Finally we see how the server and client set is done and torn down with an example.


## 4.1 Basic Concepts about User Datagram Protocol

The User Datagram Protocol (UDP) is a transport layer protocol defined for use in tandem with the IP network layer protocol. The User Datagram Protocol supports network applications that need to transport data between computers. Applications that use UDP include client/server programs like video conferencing systems. It provides a best-effort datagram service to an End System (client). UDP's main idea is to abstract network traffic in the form of *datagrams*. A datagram comprises one single "unit" of binary data; the first eight bytes of a datagram contain the *header information* and the remaining bytes contain the data itself. The simplicity of UDP reduces the overhead from using the protocol and the services may be adequate in many cases. The service provided by UDP is an unreliable service that provides no guarantees for delivery and no protection from duplication. A computer may send UDP packets without first establishing a connection to the recipient. The computer completes the appropriate fields in the UDP header (PCI) and forwards the

data together with the header for transmission by the IP network layer. The UDP header is as shown below.

## 4.2 UDP Header




**UDP header**

The UDP header consists of four fields each of 2 bytes in length

### 4.2.1 Source Port

UDP packets from a client use this as a service access point (SAP) to indicate the session on the local client that originated the packet. UDP packets from a server carry the server SAP in this field.

### 4.2.2 Destination Port

UDP packets from a client use this as a service access point (SAP) to indicate the service required from the remote server. UDP packets from a server carry the client SAP in this field.

### 4.2.3 UDP length

These are the number of bytes comprising the combined UDP header information and payload data.

### 4.2.4 UDP Checksum

A checksum is to verify that the end to end data has not been corrupted by the network or by the processing in an end system. The algorithm to compute the checksum is the Standard Internet Checksum algorithm. If this check is not required, the value of 0x0000 is placed in this field, in which case the receiver does not check the data.

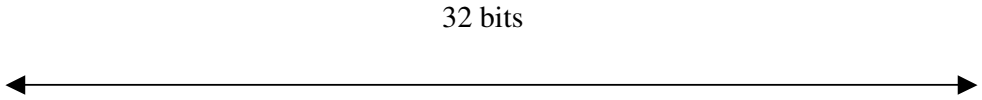
### 4.2.5 Ports

UDP utilizes ports to allow application-to-application communication. The port field is 16-bits so the valid range is 0 to 65,535. Port 0 is reserved and shouldn't be used.Ports 1 through 1023 are named "well-known" ports and on Unix-derived operating systems binding to one of these ports requires root access.Ports 1024 through 49,151 are registered ports.Ports 49,152 through 65,535 are ephemeral ports and are used as temporary ports primarily by clients when communicating to servers.

## 4.3 UDP Checksum

Like the other transport protocols, the UDP header and data are not processed by Intermediate Systems (IS) in the network, and are delivered to the final destination in the same form as originally transmitted. At the final destination, the UDP protocol layer receives packets from the IP network layer. These are checked using a checksum algorithm. This checksum works like a safety feature. The checksum value represents an encoding of the datagram data that is calculated first by the sender and later by the receiver. Should an individual datagram be tampered with or get corrupted during transmission (due to noise) the calculations of the sender and receiver will not match, and the UDP protocol will detect this error. The algorithm is not foolproof, but it is effective in many cases. In UDP, checksumming is optional; turning it off squeezes a

little extra performance from the system, as opposed to TCP where checksums are mandatory. UDP does not make any provision for error reporting if the packets are not delivered. Valid data are passed to the appropriate session layer protocol identified by the source and destination port numbers. Usually, clients set the source port number to a unique number that they choose themselves - usually based on the program that started the connection. Since the server in response returns this number, this lets the sender know which "conversation" incoming packets are to be sent to. The destination port of packets sent by the client is usually set to one of a number of well-known ports. These usually correspond to one of a number of different applications, e.g. port 23 is used for telnet, and port 80 is used for web servers.




| Source Address | | |
|---|---|---|
| Destination Address | | |
| 00000000 | Protocol Number | UDP Segment length |

Pseudo Header included in the UDP checksum

The UDP checksum field contains the UDP header, UDP data and the pseudo-header shown above. The pseudo-header contains the 32-bit IP addresses of the source and destination machines, the UDP protocol number and the byte count for the UDP segment. The pseudo-header helps to find undelivered packets or packets that arrive at the wrong address. However the pseudo-header violates the protocol hierarchy because the IP addresses, which are used in it, belong to the IP layer and not to the UDP layer.

Since UDP lacks reliability, applications must generally be willing to accept some loss, errors or duplication. Some applications such as TFTP may add rudimentary reliability mechanisms into the application layer as needed. Most often, UDP applications do not require reliability mechanisms and may even be hindered by them. Streaming media and voice over IP (VoIP) are examples of applications that often use UDP. If an application requires a high degree of reliability, TCP or erasure codes may be used instead.

Lacking any congestion avoidance and control mechanisms, network-based mechanisms are required to minimize potential congestion collapse effects of uncontrolled, high rate UDP traffic loads. In other words, since UDP senders cannot detect congestion, network-based elements such as routers using packet queueing and dropping techniques will often be the only tool available to slow down excessive UDP traffic. The Datagram Congestion Control Protocol (DCCP) is being designed as a partial solution to this potential problem by adding end host congestion control behavior to high-rate UDP streams such as streaming media.

## 4.4 UDP a connectionless protocol

UDP does just about all functions of a transport protocol.Other than multiplexing/demultiplexing function and some light error checking, it adds nothing to IP. Basically if the application user chooses UDP instead of TCP, then the application is talking almost directly with IP. UDP takes messages from application process, attaches source and destination port number fields for the multiplexing/demultiplexing service and passes the resulting "packet" to the network layer. The network layer encapsulates the packet into an IP datagram and then makes a best-effort attempt to deliver the packet to the receiving host. If the packet arrives at the receiving host, UDP uses the port numbers and the IP source and destination addresses to deliver the data in the packet to the correct application process. Note that with UDP there is no

handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be *connectionless*

## 4.5 Comparison between UDP and TCP

A common thought that would make anyone wonder is, why would any application developer use UDP over TCP. Is in not that TCP is always preferable to UDP since TCP provides a reliable data transfer service and UDP does not. The answer to this query would be no, as many applications are better suited for UDP for the following reasons:

### 4.5.1 No Connection establishment

TCP uses a three-way handshake before it starts to transfer data. UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection. This is probably the principle reason why DNS (Domain Name Server) runs over UDP rather than TCP. DNS would be much slower if it ran over TCP. HTTP uses TCP rather than UDP, since reliability is critical for Web pages with text. The TCP connection establishment delay in HTTP is an important contributor when we have to wait for the page to be loaded.

### 4.5.2 No Connection State

TCP has the characteristic of maintaining a connection state in the end systems. This connection state includes receive and send buffers, congestion control parameters, and sequence and acknowledgment number parameters. This state information is needed to implement TCP's reliable data transfer service and to provide congestion control. UDP, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

### 4.5.3 Small Segment Header Overhead

The TCP segment has 20 bytes of header overhead in every segment, whereas UDP only has 8 bytes of overhead.

### 4.5.4 Unfettered send rate

The speed at which UDP sends data is only constrained by the rate at which the application generates data, the capabilities of the source (CPU, clock rate, etc.) and the access bandwidth to the Internet. We should keep in mind, however, that the receiving host does not necessarily receive all the data .When the network is congested, a significant fraction of the UDP-transmitted data could be lost due to router buffer overflow. Thus, the receive rate is limited by network congestion even if the sending rate is not constrained.

As shown in the figure below UDP is also commonly used with multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video.

These applications can tolerate a small fraction of packet loss, so reliable data transfer is not absolutely critical for the success of the application. Furthermore, interactive real-time applications, such as Internet phone and video conferencing, react very poorly to TCP's congestion control. For these reasons, developers of multimedia applications often choose to run the applications over UDP instead of TCP. Finally, because TCP cannot be employed with multicast, multicast applications run over UDP.

| Application | Application-layer protocol | Underlying Transport Protocol |
|---|---|---|
| electronic mail | SMTP | TCP |
| remote terminal access | Telnet | TCP |
| Web | HTTP | TCP |
| file transfer | FTP | TCP |
| remote file server | NFS | Typically UDP |
| streaming multimedia | proprietary | Typically UDP |
| Internet telephony | proprietary | Typically UDP |
| Network Management | SNMP | Typically UDP |
| Routing Protocol | RIP | Typically UDP |
| Name Translation | DNS | Typically UDP |

## 4.6 Introduction to Real Time Streaming Protocol

The popularity of Internet multimedia applications has grown dramatically in the past several years, spurred by the penetration of the Web and the increasing capacity of backbone and local-access networks. Although Hyper text Transfer protocol (HTTP) can be used to transfer audio and video content, most multimedia transmissions are simply initiated on the Web. Then, the client's player contacts the multimedia server using a different set of protocols that are better suited to streaming applications. For example, the RealNetworks client and server communicate using the Real-Time Streaming Protocol (RTSP), an IETF draft standard that derived from HTTP. RTSP is used in a number of commercial streaming media applications. So it is basically a client-server multimedia presentation control protocol, designed to address the needs for efficient delivery of streamed multimedia over IP networks. The large size of most multimedia streams makes conventional Web caching techniques inappropriate. So for ease of deployment, multimedia proxy services, such as prefix caching, should not require changes to existing

client/server software or network mechanisms and should operate in the context of standard protocols. RTSP typically runs over TCP, though UDP can also be used. Though conceptually similar to HTTP, RTSP is stateful and has several different methods. The OPTIONS method inquires about server capabilities (e.g. RTSP version number, supported methods, etc.), and the DESCRIBE method inquires about the properties of a particular file (e.g. Last-Modified time and session description information, typically using SDP). Client and server state machines are created with the SETUP method, which also negotiates transport parameters (e.g. RTP over unicast UDP on a particular port). The client sends a SETUP message for each stream (e.g., audio and video) in the file. Streaming of the file is initiated with the PLAY method, which can include a Range header to control the playback point. The TEARDOWN method terminates the session, releasing the resources at the server and client sites. The protocol also includes a number of recommended or optional methods.

## 4.6.1 Other Multimedia Streaming Protocols

#### 4.6.1.1 Real-Time Transport Protocol (RTP)

RTP provides the basic functionality for transferring real-time data over packet networks. Since RTP does not include mechanisms for reliable delivery or flow control, transport of RTP packets must rely on underlying protocols such as UDP and TCP. RTP typically runs over UDP, though TCP is sometimes used for reliable transport or to stream across firewalls that discard UDP packets. RTP provides source identification (randomly chosen SSRC identifier), payload type identification (to signal the appropriate decoding and playback information to the client), sequence numbering (for ordering packets and detecting losses), and timestamping (to control the playback time and measure jitter). Data packets contain a generic RTP header with these fields, as well as payload-specific information to improve the quality of delivery for specific media (e.g. MPEG). Interpretation of some header fields, such as the

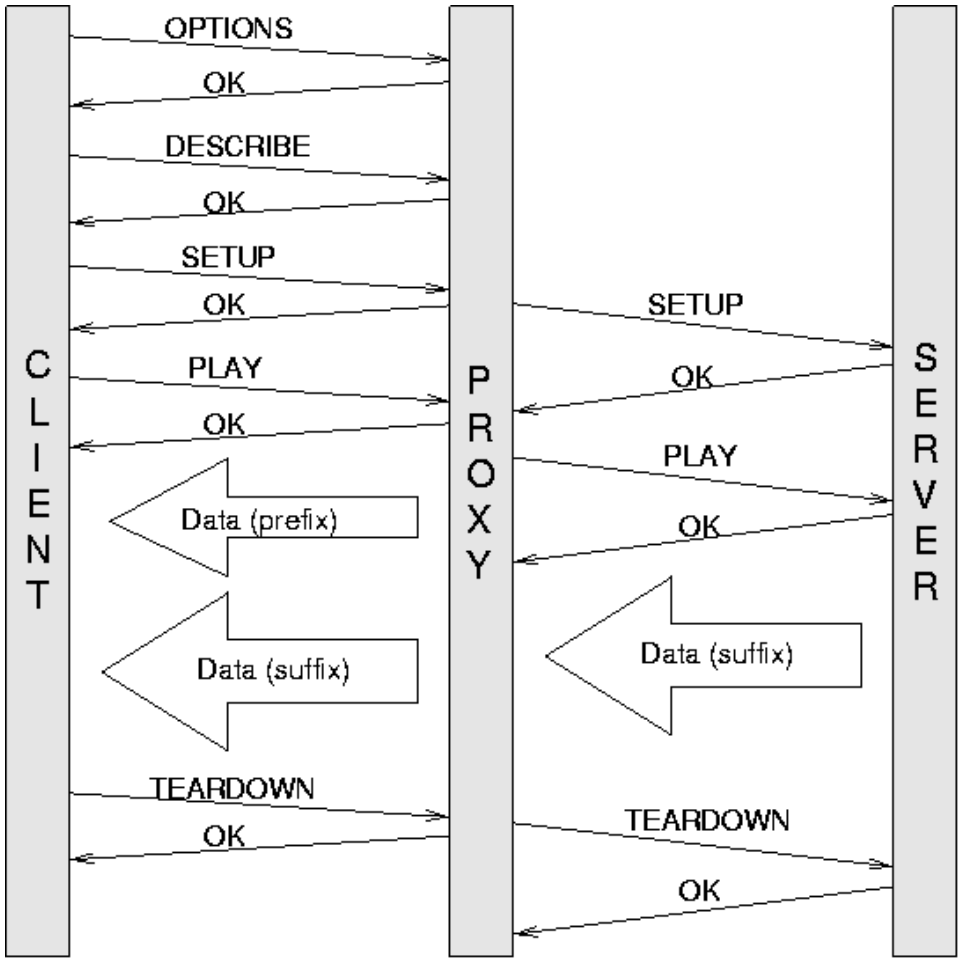
timestamp, is payload dependent. The RTP header may also identify contributing sources (CSRCs) for the payload carried in the packet; such a list is typically inserted by mixers or translators.

#### 4.6.1.2 Real-Time Control Protocol (RTCP)

RTCP monitors the delivery of RTP packets. The protocol provides feedback on the quality of data distribution, establishes an identification (CNAME) for each participant, scales the control packet transmission with the number of participants (to avoid generating excessive feedback traffic in large multicast groups), and provides minimal session control information. RTCP packets consist of source descriptors, sender reports, receiver reports, BYE packets (signifying the end of participation in a session), and APP packets (for application-specific functions). Receiver reports include the stream's SSRC, the fraction of RTP packets lost, the sequence number of the last RTP packet received, and the packet interarrival jitter. Senders can use this information to modify their transmission rates or to switch to a different encoder. The sender reports include the stream's SSRC, the sequence number of the last RTP packet sent, the wallclock time of the last transmission, and the number of RTP packets and bytes sent. The client can use the RTP timestamp and wallclock time information for media synchronization.

## 4.7 Prefix Caching in RTSP

Here it is describe how to reduce client delay by caching protocol information at the proxy, and how to use the RTSP Range header to fetch the suffix of the stream. The discussion draws on figure shown below, which illustrates the handling of RTSP messages when the proxy has cached the prefix of each stream (e.g. audio and video), the description of the presentation, and the options supported by the server.




## 4.8 Basic RTSP Operation

The handling of client requests depends on whether or not the proxy has cached the prefix and the related RTSP information. The client uses the OPTIONS message to learn which methods are supported by the server. The server (and therefore the proxy) should support the basic methods such as OPTIONS, DESCRIBE, SETUP, PLAY, and TEARDOWN. The proxy can respond directly to the client's OPTIONS request, avoiding the delay in contacting the server, as shown in figure above. In particular, the proxy could respond with a list of methods that were cached during a previous interaction with the server. Alternatively, the proxy could respond with a default list of basic methods. In either case, the proxy may occasionally send an incorrect list

of methods (e.g., if the server ceases to support some method). If the client later attempts to invoke the unsupported method, the server would simply respond with an error message (“Method not allowed") that includes the current list of methods, which the proxy can cache and forward to the client. Note that this error would not cause streaming to fail.

The DESCRIBE message is used to retrieve information about the particular presentation, which consists of one or more underlying streams (e.g. audio, video). Like the OPTIONS message, the DESCRIBE message is optional and does not affect the RTSP state machine at the client or the server. If the description resides in the cache, the proxy can respond directly to the client; otherwise, the proxy must contact the server. However, note that the cached DESCRIBE information may not be up-to-date. The proxy can employ a variety of techniques to reduce the likelihood of stale information.

The SETUP request is used to negotiate the transport parameters, including the transport protocol (e.g. RTP) and the TCP/UDP port numbers. Upon receiving the client SETUP message, port numbers are generated for the proxy's end of the RTP and RTCP connections, and a session identifier is selected. The port numbers and session identifier are sent to the client. In the meantime, the proxy generates a separate SETUP message to the server. Each stream in the multimedia presentation results in separate connections on both the proxy-client and server-proxy paths, and consists of TCP or UDP connections for RTP and RTCP. The proxy could request that the server stream the RTP packets over TCP (or a related semi-reliable transport protocol), since the cached prefix could hide transient periods of TCP congestion where the server does not send packets fast enough for continuous playback. To coordinate the transfer of RTP and RTCP messages, the proxy must maintain a mapping table to direct messages to the appropriate outgoing connection and with the appropriate session identifier.

The client can send the PLAY message after receiving the proxy response to the SETUP request. If the prefix is cached, the proxy can respond immediately

to the PLAY request and initiate the streaming of RTP and RTCP messages to the client. The client can begin playback, without waiting for the proxy to communicate with the server. Note that the proxy may or may not have received the SETUP response from the server, depending on the delay on the server-proxy path. Once the proxy has received a SETUP response from the server, indicating the server's session identifier, the proxy can send the PLAY request to the server, with the appropriate Range header. After transmitting the prefix of the stream to the client, the proxy can start sending the RTP packets in the suffix retrieved from the server.

Depending on the size of the prefix, the proxy may decide to wait longer before asking the server to transmit the suffix. Fetching the suffix too early would require extra storage resources at the proxy, and may result in extra server load and network traffic if the client eventually pauses or stops the transfer of the presentation. Yet, the suffix should be requested sufficiently ahead of when the proxy would need to start transmitting this data to the client. Requesting the suffix a bit early allows the proxy to hide the server-proxy round-trip delay (which can be estimated from RTCP reports), or additional delays for other functions such as smoothing or transcoding.

## 4.9 Linking the Prefix and Suffix

When the entire stream resides in the cache, the proxy acts as a server in responding to the client PLAY request. When the cache does not contain any portion of the requested resource, the proxy forwards the PLAY request to the server and simply shuttles messages back and forth between the server and the client, acting as an application-level router. The operation of the proxy becomes more interesting when the cache stores only the prefix of the stream. In this case, the proxy can reply to the client PLAY request and initiate transmission of RTP and RTCP messages to the client, while requesting the suffix from the server. Fetching the suffix requires the proxy to initiate a Range request for the appropriate portion of the stream. As part of linking the

prefix and suffix, the proxy must specify the appropriate Range and handle inaccuracies in the server response.

As part of caching multimedia content, the proxy keeps track of the size of the prefix in terms of the timestamps in the RTP packets. The RTSP Range request is defined for both SMPTE Relative timestamps and NPT (Normal Play Time). The proxy does not know in advance whether or not the server supports Range requests for a particular presentation (some streams do not allow seek operations), and whether or not the SMPTE format is supported. The proxy can learn, and cache, this information for each presentation by sending Range requests to the server and noting the response (e.g. the server sends a 426 ``Header Field Not Valid for Resource'' response when Range is not supported). The proxy could avoid polling the server for this information if the RTSP SETUP or PLAY response included information about support for Range requests. To avoid changing existing RTSP implementations, providing the information about Range support could be optional, since the proxy could always infer the information and/or decline to cache partial contents when the information is not provided.

In addition, RTSP does not require the server to handle the Range request precisely. For example, the request may identify a starting time that does not correspond to any particular frame, forcing the server to initiate transmission from the previous or subsequent frame. In addition, the server may not support arbitrary indexing to individual frames. (e. g. an MPEG stream typically consists of I, P, and B frames within a group of pictures (GOP). The server may not precisely satisfy a Range request that starts in the middle of a GOP. In fact, allowing clients to index to any arbitrary frame could introduce substantial overhead at the server (to maintain fine-grain indices, or to parse the content to sequence to the appropriate frame), particularly for variable-bit-rate streams, or streams without a fixed GOP structure.) The RTSP protocol does not dictate how accurately the server must handle the Range header, and does not provide a way for the server to indicate in advance how much

inaccuracy could be introduced. As such, the proxy should be conservative, and ensure that the transmission to the client does not have duplicate or missing packets. Fortunately, the server reply includes a Range response header that indicates what range of time is currently being played. If the beginning of the suffix contains frames that are already at the end of the prefix, the proxy discards the overlapping packets. To avoid a gap between the prefix and the suffix, the proxy could initiate a conservative Range request, or issue a second request if a gap arises.

## 4.10 Seamless Transmission of RTP Packets

In order to link the prefix and the suffix, all RTP headers must be consistent; otherwise the client will not associate the two parts to the same stream. The sensitive fields are sequence numbers, timestamps, and source identifier (SSRC), which have been selected separately by the proxy (for the prefix) and the server (for the suffix). Therefore, the proxy will have to change the RTP header fields of the suffix to match the SSRC it chose for the prefix, and the timestamps and sequence numbers to indicate that the suffix must be played after the prefix. In streaming the suffix, the proxy overwrites the SSRC field in each RTP packet with the value it selected as part of initiating transmission of the prefix. The proxy knows the timestamp and sequence number used in transmitting the last RTP packet of the prefix. The base timestamp and sequence number for the server's transmission of the suffix are provided in the RTP-Info header in the PLAY response. The proxy can then add/subtract the appropriate constant for the timestamp and sequence number fields of each packet in the suffix.

## 4.11 Decoupling of Client and Server

Prefix caching decouples the server transmission from client reception. Caching partial contents of the stream, and overlapping the prefix transmission on the proxy-client path with the suffix on the server-proxy path, introduces

new challenges in cache coherency and feedback control. This section identifies the key issues and presents several possible solutions.

## 4.12 Cache Coherency

Caching RTSP messages and multimedia content at a proxy introduces potential coherency problems when the server changes this information. Cache coherency is a critical issue in the Web, where many resources change very frequently. The problem is arguably less important for multimedia presentations, which may change less frequently - the server could simply assign a new URL when new content is created. Still, the paradigms for authoring and updating multimedia content are not well understood, and the use of proxies should not place additional restrictions on how servers are managed. As such, the proxy should incorporate mechanisms to prevent the transmission of a suffix that does not come from the same version of the presentation as the cached prefix.

## 4.13 Prefix Caching Implementation

- Client RTSP connection: Upon establishing a TCP connection with the client, the proxy records the client IP address and the identifier for the TCP socket.
- First client RTSP request: The first RTSP message could be an optional OPTION or DESCRIBE message, or a SETUP message. The proxy extracts the server name and performs a gethostbyname () to learn the server IP address. The proxy establishes a TCP connection to the server and stores the identifier for the TCP socket.
- Client SETUP request: The client SETUP message(s) includes information about the transport protocol and port numbers that the client wishes to use. After binding the two UDP sockets, the proxy records the two sets of client and proxy port numbers, and replies to the client. In the meantime, the proxy generates and records the port

numbers for its UDP connections with the server, and sends a SETUP message to the server.

- Server SETUP response: After receiving the server reply, the proxy records the server port numbers for the UDP connections and binds the UDP sockets. The proxy also stores the session identifier, generated by the server.
- Client TEARDOWN request: The proxy closes the sockets to the client, forwards the TEARDOWN to the server, and deletes the session data structure.

# Chapter 5

# The System Design


## Summary

This chapter discusses the entire system design and the tools used in implementing this system. The FFMPEG tool is discussed in detail with the a sample configuration file. We conclude this chapter with some experimental results from the codes that we have implemented.


## 5.1 Video Input

The input to the system is a video captured by a camera. The camera that is being used is an Apple iSight camera. It's a state of the art video camera featuring an autofocusing autoexposure F/2.8 lens that captures high-quality pictures even in low lighting. A single FireWire cable streams video and audio and also delivers power to the camera. A custom-designed, three-part lens consists of two aspherical elements that focus on a 1/4- inch CCD sensor with 640x480 (VGA) resolution. The camera's lens aperture is a wide F/2.8, allowing it to collect more light.

iSight delivers smooth, top-quality, full-motion video at 30 frames per second in 24-bit color.

The video captured from the camera is then given to the FFMPEG a tool configured for video encoding. This video is in a RAW video format called as YUV 422 format. The size of any RAW video is quite large. FFMPEG compresses this RAW video into a video compression standard called H.263. The size of the RAW video is reduced drastically which is why the H.263 standard is an a compression algorithm defined for low bitrates.

| iSight Technical Specifications | |
|---|---|
| Requirements | Requires 600MHz G3 processor or higher |
| Sensor | 1/4-inch color CCD image sensor, 640x480 VGA resolution |
| Focus | Autofocus from 50mm to Infinity |
| Framerate | Full motion video at up to 30 frames per second (FPS) |
| Input and Output | FireWire for audio, video and power connection |
| Audio | Integrated, dual-element microphone with noise suppression |

## 5.2 Video Compression

### 5.2.1 FFMPEG

FFMPEG is a very fast video and audio converter, which can also grab from a live audio/video source. It is an open source being developed regularly by experts from the field of video codecs. It has a very friendly command line interface, which is innate such that it tries to build all parameters automatically. We normally have to specify target bit rate. FFMPEG is basically a complete elucidation to record, convert and stream audio and video. It includes **libavcodec**, the leading audio/video codec library. FFMPEG is developed under Linux Operating System.

The FFMPEG is composed of several components

- FFMPEG which is a command line tool to convert various video formats. It supports grabbling live video from web cameras or even TV card.

- FFSERVER which is one of the main components of this programme is an HTTP server (multimedia streaming also supported) for broadcasting.
- FFPLAY is a simple media player based on libraries.
- LIBAVCODEC is an open source library containing encoders and decoders for FFMPEG.
- LIBAVFORMAT is a library containing parsers.

FFMPEG can use a video4linux compatible video source.
Note that you must activate the right video source and channel before launching FFMPEG.
Example
ffmpeg /tmp/out.mpg

There are several things one has to remember before launching FFMPEG.
One most important thing is to keep in mind that ffmpeg can use only supported file format and protocol as input.
Example
We can input from YUV files by using the command:-
ffmpeg -i /tmp/test%d.Y /tmp/out.mpg
As we know that Y files use twice the resolution of the U and V files. Y files are raw files, without header. All decent video decoders can generate them. We must specify the size of the image with the option
{-s} option if ffmpeg cannot recognize it.

### 5.2.2 Syntax

The Generic Syntax is: -
ffmpeg [[infile options][`-i' *infile*]]... {[outfile options] *outfile*}...

As a general rule, options are applied to the next specified file.
For example, if you give the `-b 64' option, it sets the video bitrate of the next file. The format option may be needed for raw input files.

By default, FFmpeg tries to convert as losslessly as possible: It uses the same audio and video parameters for the outputs as the one specified for the inputs.

## 5.3 FFserver

This is an IP based streaming server. As we know that streaming services require a certain amount of bandwidth to ensure the bit-rate needed by each media stream and the strict delay variation (i.e. jitter) needed to avoid buffer underflow at streaming clients. The FFserver is designed taking into account these considerations.
The FFMPEG is used to send live feeds to FFserver. When FFserver detects that a live feed has been sent, it then starts the streaming services.
Some specifications like

- The port on which the server is listening.
- Address on which the server is bound.
- Simultaneous requests that can be handled.
- Maximum amount of data rate that could be consumed when streaming to many clients.

have to be known to FFserver before it can start streaming.
Before starting the FFserver an appropriate path must be defined for FFMPEG. Each stream must also be defined which will be generated from the original video stream. E.g. test.mpg. FFserver will send this stream when answering a request for this file name.
A configuration file has to be configured according to the requirements of a client. In the configuration file all the other details of the stream must be defined. These include

- Format for the stream (MPEG,RM,AVI)
- Bitrate for the video stream

- Rate control buffer
- Number of frames per second
- Size of the video frame (e.g. 720 x 480 or 720 x 576)
- Transmission of only Intra Frames which is very useful for low bitrates
- If we need a suppressed video

A sample configuration file is as shown below:

Port on which the server is listening.
# 8090

Address on which the server is bound.
# 194.90.224.35

FFMPEG should send a live feed to FFserver.
# FFMPEG http://localhost:8090/feed1.ffm
Define each stream, which will be generated from the original video stream.
<Stream test1.rm>
coming from live feed 'feed1'
Feed feed1.ffm
mpeg : MPEG-1 multiplexed video and audio.
rm : Real Networks-compatible stream. Multiplexed audio and video.
avi : AVI format (MPEG-4 video, MPEG audio sound).

Bitrate for the video stream
# VideoBitRate 200

Ratecontrol buffer size
# VideoBufferSize 40

Number of frames per second

# VideoFrameRate 15

Size of the video frame

# VideoSize 720x480

Size of the Group of Pictures

# VideoGopSize 12

Example streams

Real video at 64 kbits

<Stream test.rm>

Feed feed1.ffm

Format rm

VideoBitRate 128

VideoFrameRate 25

VideoGopSize 25

NoAudio

## 5.4 FFplay

FFplay is a simple media player designed using the FFMPEG and SDL libraries.

The command line interface would be

ffplay –f rm http://merkur80:8090/test.rm

Different options are available when the video is playing,

q----quit

f----fullscreen

p----pause

## 5.4 Transmission Protocol

When we think of transmission protocols Transport Control Protocol (TCP) seems to be the obvious choice due to its reliable and wide spread applications. In real time video transmission the three most important things to be taken into consideration are

- Time Delay
- Bandwidth
- Jitter

Due to these three reasons traditional video applications have shied away from TCP and have utilized UDP with greater flexibility. Only one big concern of using UDP is that it does not support guaranteed transfer. So this gives us more flexibility in our error protection.

In our system we use UDP as our transport protocol between the server and the client. The H.263 encoded stream, which is generated using FFMPEG, is then error protected with Digital Fountain Codes as explained above and then sent over the network using UDP protocol.

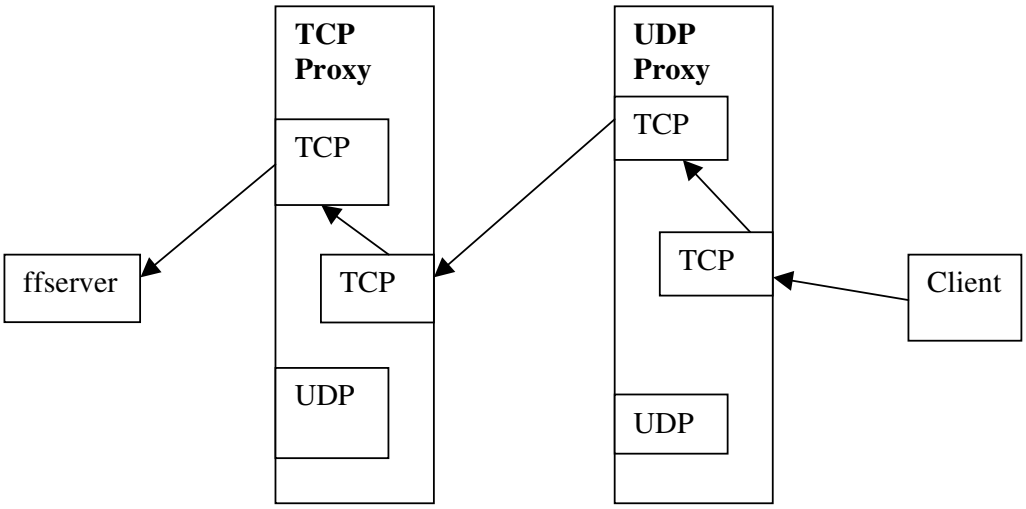
At the decoder the reverse procedure takes place. First the received stream is decoded using Digital Fountain Codes and then the H.263 decoding algorithm is run to receive the original source stream.

When client wants to view the video we type in the request

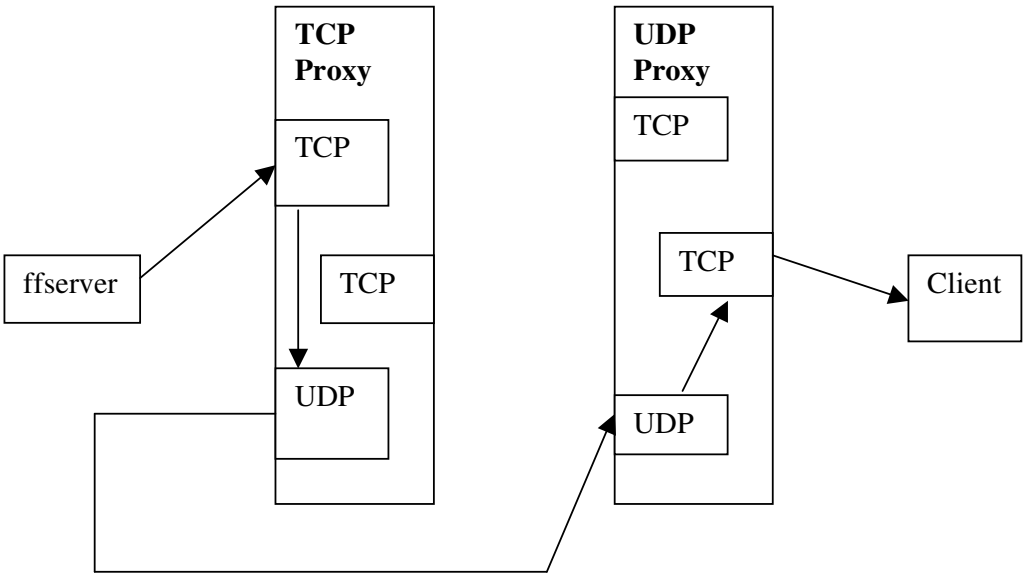
http://localhost:8080/ test.rm

As we can see we are requesting for a TCP request. So how is it that we propose for a UDP transmission? The system is designed in such a way that there two proxies set up between the FFserver and the client. So basically there exists a server proxy at the server side and a client proxy at the client side. The figure is as below.

Client --- TCP --- ClientProxy -- UDP -- TCP -- ServerProxy -- TCP -- Server




As we can see, the client requests for the video stream from the server. This request has to be a reliable one, which guarantees that the request reaches the server. Hence we have set it up as a TCP request. Once the server has a request from the client the streaming starts. This can be explained as below.



The server then sends a TCP stream to the TCP port of the TCP proxy. At this point the UDP port is enabled and video stream is then sent to a UDP port of the TCP proxy.

The command line interface to start the proxy is as below
./proxy [udp, tcp] listenTCPPort ip writeTCPPort rwUDPPort

[udp,tcp] – Use udp for the ClientProxy, tcp for the ServerProxy listenTCPPort - on this port, ClientProxy waits for connection from the client; ServerProxy waits for connection from the ClientProxy. ip and writeTCPPort - ClientProxy: ip and port of the ServerProxy (port is the listenTCPPort); ServerProxy: ip and port of the Server rwUDPPort - ClientProxy reads on this port, ServerProxy where it sends to should be identical.

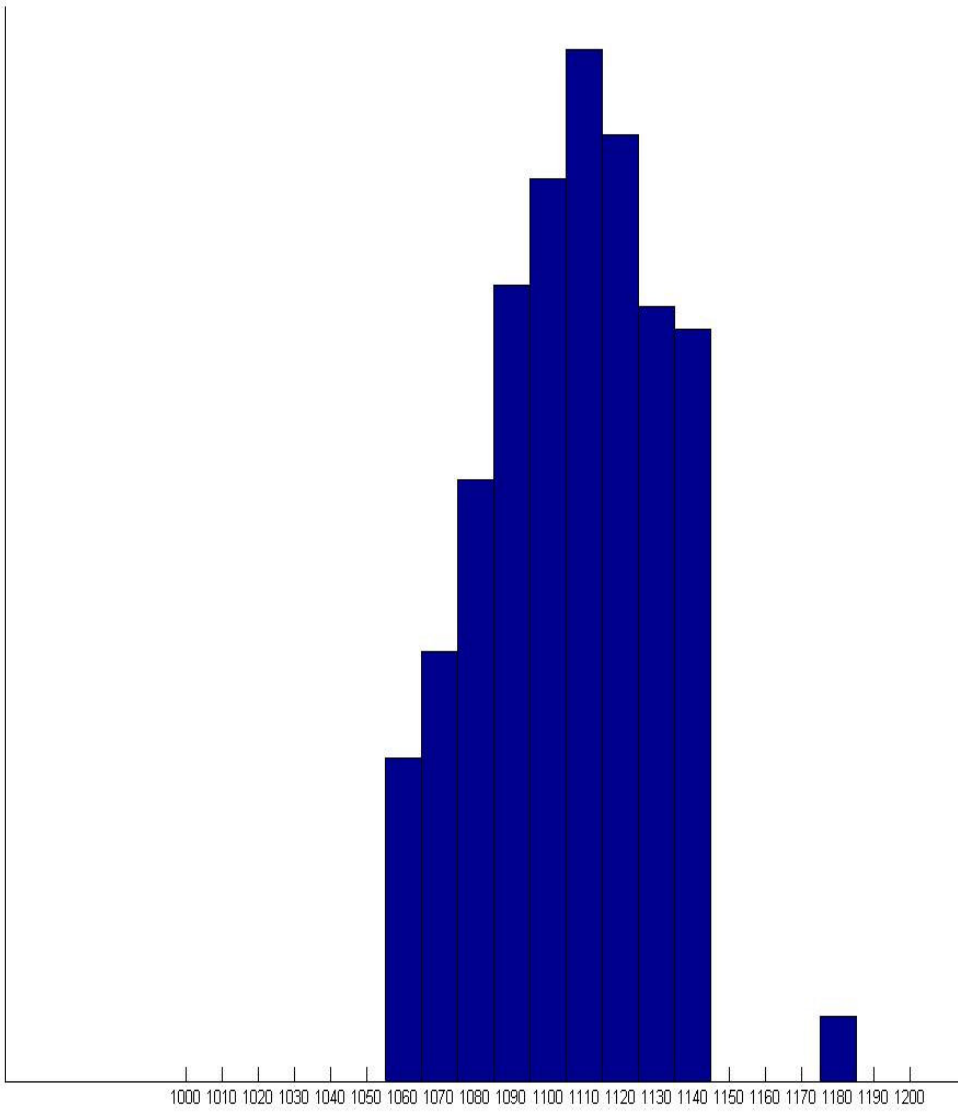
Example:-
./proxy 8090 merkur80 8070 7000
The TCP proxy, which is the SERVER proxy, is started now.
So basically we are now sending UDP packets over the network.
At the clients side we have to start the client proxy to receive the incoming data (video stream). This is done as below

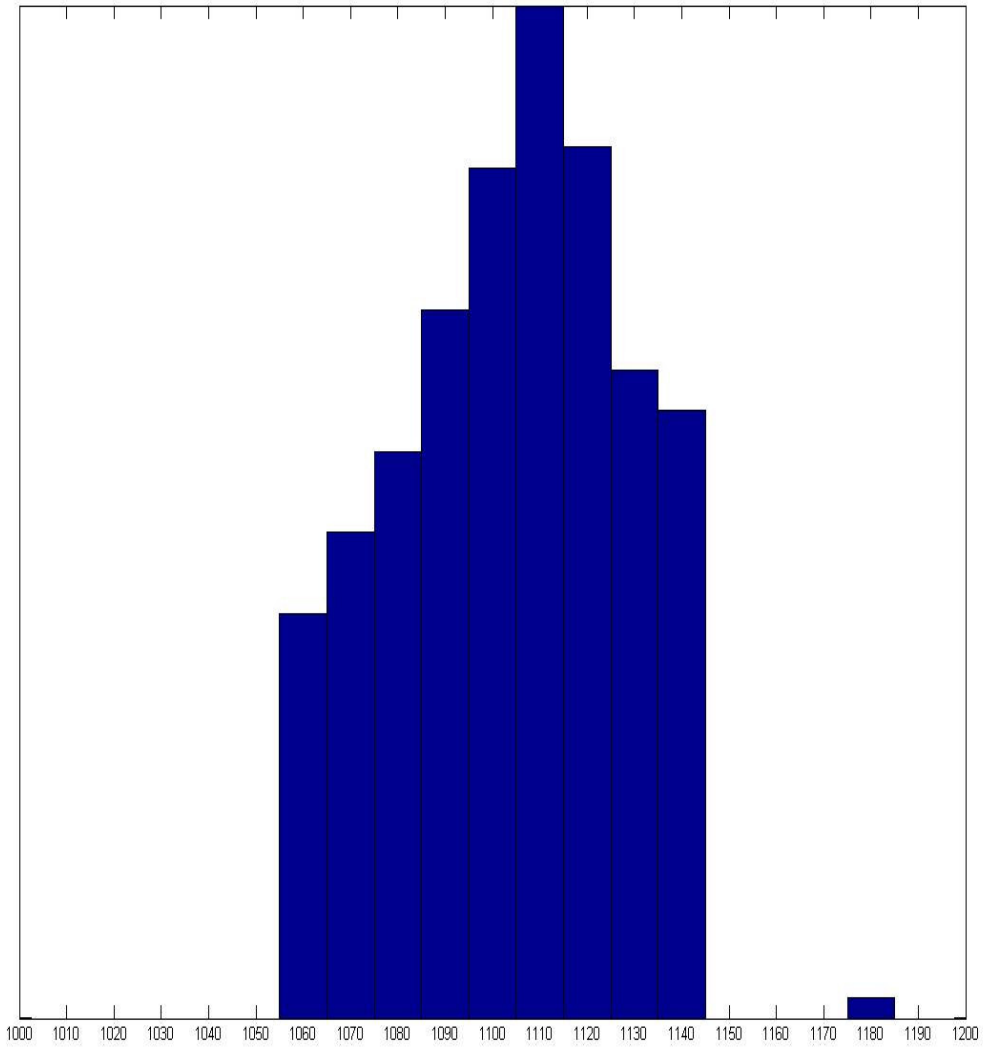
./proxy udp 8080 localhost 8090 7000

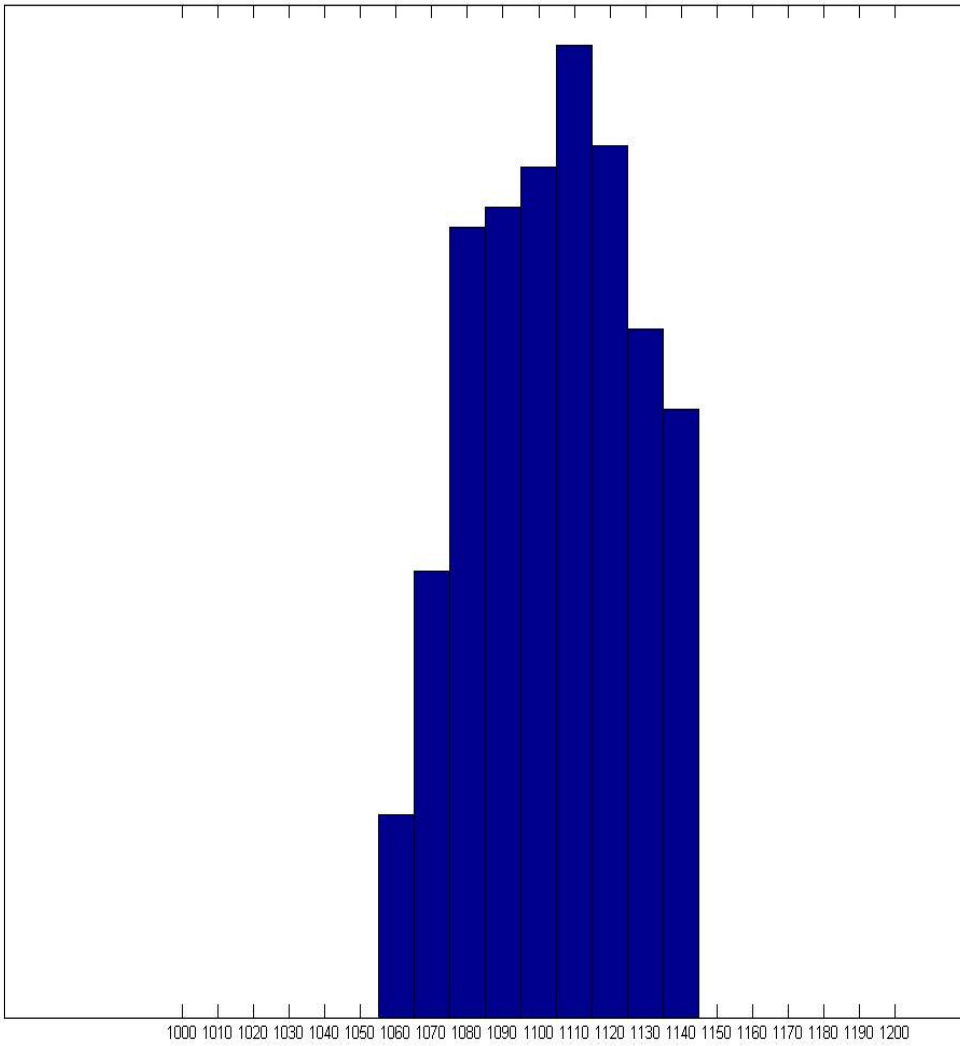
The UDP packets are received on port 7000 and then this UDP data is converted back to TCP and sent as TCP data to the client's media player, which he uses to view the video stream.

This is how the proxies work and we achieve UDP transmission over the network.

## 5.6 Experimental Results for Luby Transform Codes




1000
1010
1020
1030
1040
1050
1060
1070
1080
1090
1100
1110
1120
1130
1140
1150
1160
1170
1180
1190
1200


1000 1010 1020 1030 1040 1050 1060 1070 1080 1090 1100 1110 1120 1130 1140 1150 1160 1170 1180 1190 1200

# Chapter 6

## Summary

In this chapter we conclude our thesis and discuss the future scope by seeing some applications where this system would be handy.

# Conclusion and Future Scope

In this thesis, we presented a new approach for scalable video coding with simple on the fly error protecting codes. We initially motivated our study of scalable video by examining the intended applications. It was noted that for communication over heterogeneous channels, scalable video permits users to receive the highest quality video that their channel can support. Additional applications included public safety or surveillance for which high scalability allows low bitrate transmission and local storage of the high quality sequence. A final application was communication across lossy channels where scalable bit stream protected with Luby Transform Codes allowed an error resilient representation.

## 6.1 Applications

Once consumers get access to high-bandwidth solutions like cable and ADSL, we can finally get rid of all those hokey 28.8K and 56K modems. Right now, bandwidth is the biggest problem facing companies and it is preventing many more companies from putting more streaming content on the Web. Although there's a lot of content available right now, companies are still staying away. Watching a 4-inch window at 10 frames per second gets tiring very quickly, that's why most videos online are just a few minutes long. In the near future, content developers and site administrators will need extra bandwidth to carry the large load of audio and video. Also any provider working with IP

multicasting will also play a major role. Companies will have to handle the encoding, networking, servers, and media delivery themselves or hire outside help to do it for them. In the short-term, companies will flock to streaming media developers, but networks and TV stations will eventually catch up and do their own streaming delivery themselves.

Right now the biggest application for streaming media in a corporate intranet is training and voice conferencing, which saves the transit time of the employees inside the premises. Employees can just sit on their desks and have live instruction or on-demand access to training videos. Besides training, other applications include news delivery, watching company addresses and meetings, video monitoring applications, and more.

And a concept of "Remote Troubleshooting" or "Troubleshooting from/on your Desktop" can be easily attained using the Video conferencing on a Cable Network. The employees can interact directly with any other employee from his/her own desktop PC or Laptop, as if "in person". Streaming technology thus proves to be the most promising solution for this today, with high data rate up to several Mbps and at the same time very economical. Further, the most recent enhanced streaming standard can give up to several Mbps data rate, which further broadens its utility.

Eventually, streaming video networks will be very commonplace for businesses, and we'll all wonder how we got along without them.

With further development this project can be extended to be used as:

### 6.1.1 Anti-Theft Video Security Systems

With the unexpected rise in thefts with culprits striking places like banks, jewelers, malls etc, which now a days use a wired network of cameras manned at regular intervals, a streaming system like this can be an excellent

replacement, as the whole security system becomes secure and the cameras units can be placed at any point as desired.

To achieve this project can be modified such that it will consist of a Video Camera directly interfaced to the monitoring system which enables Real Time video steaming. Additionally, motion sensors, smoke detectors etc. can be incorporated into this system. These camera units can act as independent units, all sending continuous streaming video to a central system which is manned.

### 6.1.2 Remote Video Surveillance Systems

Many corporations and utilities have an issue of how to control access to remote, unmanned buildings. This design can also be mounted on surveillance vehicles, which will be used to survey hazardous and unfamiliar territories. Thus this system can be in integral part of equipment like Spy Planes, Surveillance Robots etc. The potential costs associated with staffing such premises to ensure visual verification can be highly restrictive in assuring a secure managed entry system. In many instances keys and swipe card systems suffice, but for true security with flexibility, the use of remote video transmission technology to control access from a central receptionist is unbeatable.

### 6.1.3 Compressed Video (MPEG4) Enhancement

Using DSP Codec Chips to compress the audio and video data traveling over the network can further enhance the project. The latest trend these days are the High Speed MPEG4 DSP Chips, which directly interface with a video camera and can compress raw video on the fly. These codec chips shrink video by replacing the original frames with more compact versions. And with the

continuous increase in the processing speeds of these chips, real time video compression is becoming better and better. Decoders or players, decompress the shrunken files and play them back as audiovisual files. Thus we can send a much better quality video over a limited bandwidth.

The solution to the bandwidth problem is faster Internet connection options, such as DSL, cable, and broadband and the latest being Wi-Fi. Despite all of the media hype that these faster services are popping up everywhere, at this time they simply are not widely available in most areas of the country. The demand for higher bandwidth is growing faster than telecommunications companies can provide faster pipelines. Competition to provide faster Internet service is extremely hot, and today, when looking for a streaming media solution, both developers and service providers are particularly looking into such a system as it is the most promising solution to the problem of limited bandwidth, as faced in traditional RF communication.

# BIBLIOGRAPHY


[1] Information Theory, Inference, and Learning Algorithms V.Bhaskaran and K.Konstantindies Image and Video Compression Standards: Algorithms and Architecture, Boston, Massachusetts.

[2] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik, "Video Coding for Streaming Media Delivery on the Internet," *IEEE Trans. Circuits and Systems for Video Technology*, March 2001.

[3] Y. Wang, J. Ostermann, and Y. Q. Zhang, *Video Processing and Communications,* New Jersey, Prentice-Hall, 2002.

[4] M.-T. Sun and A. Reibman, *Compressed Video over Networks*, Marcel Dekker, New York, 2001.

[5] M. Flierl, T. Wiegand, and B. Girod, "A Locally Optimal Design Algorithm for Block-Based Multi-Hypothesis Motion-Compensated Prediction", in Data Compression Conference, Snowbird, USA, Mar. 1998.

[6] Information Theory, Interference and Learning Algorithms: David J.C. MacKay, Version 7.2 (fourth printing) March 28, 2005.

[7] Draft ITU-T Recommendation H.263. Video coding for Low bitrate communication, May 1996.

[8] Jarno K. Tanskanen, Tero Sihvo, and Jarkko Niittylahti: H.263 Video Encoder Implementation., Tampere, Finland.

[9] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," IEEE Trans, Dec. 1981.

[10] M. Luby, \LT- codes," Proceedings of the 43rd Annual IEEE Symposium on the Foundations of Computer Science (STOC), 2002

[11] "Video coding for low bit rate communication", ITU-T Rec. H.263, International Telecommunication Union, version 1, 1996; version 2, 1997.

[12] http://www.apple.com/isight/

[13] http://www.ietf.org/rfc/rfc0768.txt

[14] Overview of Networking Tutorial, Custom Networking, Sun Microsystems

[15] Lars-Åke Larzon, Mikael Degermark, and Stephen Pink: UDP Lite for Real Time Multimedia Applications, Extended Enterprise Laboratory, April 1999.

[16] Salman A. Baset and Henning Schulzrinne Department of Computer Science Columbia University, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, September 2004.

[17] Randall Stewart, Paul D. Amer: Why is SCTP needed when TCP and UDP are widely available? ISOC MEMBER BRIEFING #17, June 2004.

[18] Edward Criscuolo**:** Performance Efficiency of Internet Protocol (IP) in Space Applications, Computer Sciences Corp. for the Goddard Space Flight Center, November 1999.

[19] http://ffmpeg.mplayerhq.hu/

[20] Jerry Gibson, Toby Berger, Tom Lookabaugh, Dave Lindberg, Richard Baker: Digital Compression for Multimedia 1998.

[21] Sayood K: Introduction to data compression, San Francisco 1996.

[22] http://www.rtsp.org/

[23] Schulzrinne H., Casner S., Fredrick R and Jacobson V. RFC 1889: A Transport Protocol for Real Time Applications.

[24] K. Rao and J. Hwang, 1996: Techniques and Standards for Image, Video and Audio Coding.