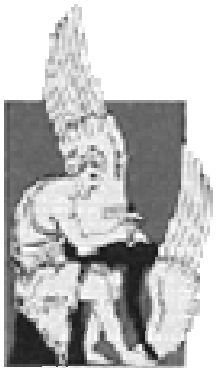
TECHNICAL UNIVERSITY OF CRETE

Department of Electronic and Computer Engineering



DIPLOMA THESIS

REAL - TIME TRANSMISSION OF VIDEO WITH ERROR CORRECTION ALGORITHM

TSAGKATAKIS GRIGORIS

Supervising committee: Zervakis Michalis(Supervisor)
Petrakis Euripides
Liavas Athanasios

Chania 2005

Acknowledgements

I would like to express my gratitude to all the people that helped me during my studies.

First of all my parents for their support in every decision I make and action I take. Their qualities as scientists and above all human have always been a guideline for me.

To my teachers, Michalis Zervakis and Euripides Petrakis for their time, their ideas and their valuable advices.

To professor Athanasios Liavas for his support in my diploma thesis.

To all my friends, for being there when I needed them…

“A wise man can see more from the bottom of a well than a fool can from a mountain top”.

Anonym

Table of Contents

Introduction

The real-time transmission of video over the Internet is one of the most demanding applications in terms of bandwidth and processing power. Yet, it is becoming an important building block of numerous applications, such as Internet television, video conferencing, distance learning, digital libraries, tele-presence and video-on-demand. The main requirements are enough bandwidth, minimum delay and small loss packet rate. However the Internet does not propose any mechanism for Quality of Service (QoS) guarantee. In the case of multicast video transmission the heterogeneity inherited in the Internet creates more problems is terms of bandwidth efficiency and service flexibility.

The challenging QoS criteria are as follow:

- Bandwidth: In order to receive a good quality video, in terms of frame rate and Signal to Noise ratio, a minimum bandwidth needs to be set. Unfortunately the Internet does not provide any means of bandwidth reservation, at least at its current stare. Furthermore, typical routers do no provide any congestion control and excessive traffic can lead to a congestion collapse, further reducing the quality of the receive video.
- Delay: In order to keep up with the real time requirement of the transmitted video, a maximum end-to-end delay must be set and secured throughout the transmission of the video. This means that any packet send through the Internet must be received, processed and displayed in time or else it becomes useless. In that cause the playout process will be paused, and valuable information will be lost. Although the real time transmission o video requires a small and constant delay, Internet does not provide any mechanism to preserve the delay. In particular, in the case of excessive traffic, the delay could become highly variable.
- Loss: Packet loss can become a major reason for the degradation of the video quality. Fortunately, due to the temporal redundancy of the images that make the video, there are various algorithms to minimize the effect of a relatively small amount to packet loss. Although real time video has a maximum packet loss requirement, Internet cannot provide any packet loss threshold guarantee.

There are many video compression algorithms such as MPEG-2, H263 and H264 (MPEG-4), who can achieve very good results due to the utilization of both temporal

and spatial redundancy in the video. Nevertheless MJPEG compression algorithm is chosen for our application. MJPEG compression algorithm compresses each frame independently of any other by taking advantage of the spatial redundancy of the frame. Frames are then displayed in an orderly fashion at a certain frame rate thus creating the illusion of continuous video playback. MJPEG does not achieve as good results as the MPEG-2, the H263 or the H264 (MPEG-4). Nevertheless its computation complexity makes it preferable for our application. This choice is made more apparent by considering the qualities of the transmitted video.

Our application's main purpose is the transmission of surveillance video. There are two main qualities that distinguish the surveillance video from any generic video
In surveillance video, there is usually little or no movement and the video is useful as long as it is acquired, transmitted and displayed with minimum delay.

Because of the little movement present is surveillance video, successive frames tend to be very similar. This temporal redundancy property is used in order to conceal the errors created by the loss of packets during the transmission of the video. On the other hand, minimising the delay of the transmission makes more complex algorithms ill suited for our applications. More complex, yet more efficient compression algorithms, introduce a significant delay due to their computational complexity
In addition, the lack of any guaranty about the available bandwidth between the server and the client may result in the loss of packets and therefore the loss of a frame. A special care must be taken so that the loss of a packet will not result in the lost of the frame, and that the variations of the available bandwidth will not cause a significant degradation of the receiving video quality.

Application's objectives

The objective of our application is two fold. First, we implemented a protocol for transmitting MJPEG video over packet-switched network such as the Internet and second, we implemented an error concealment algorithm in order to increase the quality of the received video, even in the cases of major packet loss.

The transmission system consists of one server transmitting the video and one or more clients receiving the video. Our MJPEG video transmission algorithm is build over the UDP transport protocol. The UDP transport protocol in responsible for transmitting packets (datagrams). The server is responsible for placing the MJPEG video bitstream into fixed size packets and transmit the over the network. The client on the other side is responsible of putting the packets in the correct order, recreating the frame and displaying the video.

Unfortunately, UDP protocol does not guarantee the correct arrival of the packets neither the correct ordering of them. Therefore, our application is responsible for estimating the missing packets and correctly reordering the packets. The estimatation of the missing packets is performed based on the temporal redundancy of the video. If a packet is loss, the corresponding packet from the previous frame is used to fill in the gap. The packet reordering is achieved by examining the header of each received packet and placing it to its corresponding position into a temporary buffer, until the final packet is received.

Organization of the thesis

In chapter 1, we present the JPEG still image compression standard.
In chapter 2, we make a brief introduction to the protocols used in the Internet
In chapter 3, we present our implementation
In chapter 4, we give the experimental results of our algorithm
In chapter 5, we describe the implementation system and discuss the conclusions and further improvement.

CHAPTER 1

JPEG Compression Algorithm

JPEG is the most widely used standard in image compression today. JPEG stands for Joint Photographic Experts Group and was developed by ISO and CCITT. JPEG can compress both grayscale and colored images in a lossless or a lossy way.

1.1. Modes of operation

The JPEG standard defines four modes of operation:

I. Sequential encoding, where the image, in the case of grayscale images, or a component of the image, in the case of color image, are encoding in a single left-to-right, top-to-bottom scan.

II. Progressive encoding, where the image is encoding in performed in multiple scans, intentioned for applications where transmission in long, and the viewer prefers to watch the image build-up in multiple coarse-to-clear passes

III. Lossless encoding, where the image encoded is guaranteed to be identical to the original image

IV. Hierarchical encoding, where the image is encoding in different resolution, so that lower-resolution versions may be accessed without first having to decompress the image at its full resolution

1.2. Compression

The Sequential encoding, where the image is encoding from left to right and from top to bottom, the Progressive encoding where the image is encoding in multiple scans and the compressed image is build up, the lossless encoding where the encoded image is guaranteed to be the same as the original one and the Hierarchical encoding, where the image is encoding in multiple resolutions giving the option of decompressing it at a lower resolution.

The encoding procedure of a grayscale image is illustrate in figure 1.1

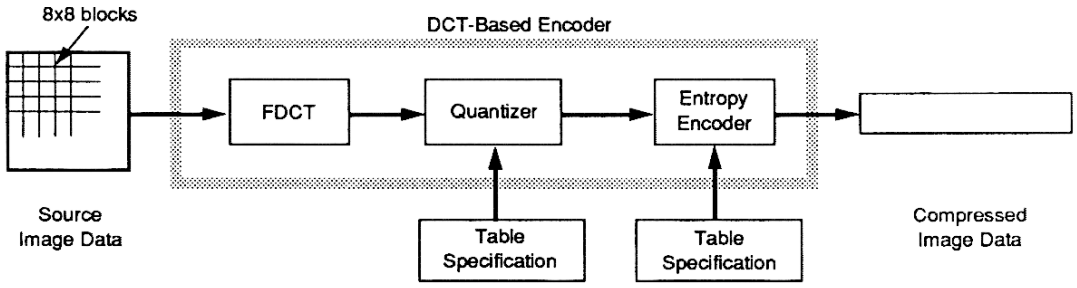


Figure 1-1

1.2.1. DCT

The matrix representing the image is grouped into 8x8 blocks and shifted from unsigned integer with range [0, 2p – 1] to signed integers with range [-2p, 2p-1]. Those blocks are feed into the Forward Discrete Cosine Transform (FDCT).

The following equation describes the FDCT.

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * \cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right]$$

Where: $C(u), C(v) = 1/\sqrt{2}$ for u,v=0

1 otherwise

Each 8x8 block is transformed into a 64-point discrete signal as a function of the two spatial coordinates x and y. The DCT coefficient with zero frequency in both x and y is called the “DC coefficient” and the remaining 63 are called “AC coefficients”.

1.2.2. Quantization

The output of the DCT encoder is feed into a uniform quantizer in accordance with a 64-element Quantization Table, which is defined by the application. In general Quantization is the process of dividing each DCT coefficient by its corresponding quantizer step and rounding it to the nearest integer, following the equation

$$F^{Q}(u,v) = IntegerRound\left(\frac{F(u,v)}{Q(u,v)}\right)$$

After the quantization is the process of DC coefficient encoding and “zig-zag”. In this step the DC coefficient, which is a measure of the average value of the 64 image

samples of the 8x8 block, is encoded as the deference from the previous DC coefficient, i.e.

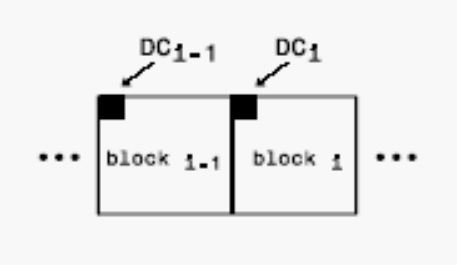


Figure 1-2

This is done because of the high correlation of the DC coefficients between adjacent 8x8 blocks and the fact that DC coefficients frequency usually contains a large portion of the image's energy.

The DCT coefficients are ordered in a "zig-zag" way as show in figure 1-3. This is done because it places the low-frequency coefficients (most likely to be non-zero) before high frequency coefficients, increasing the performance of the next step which is the entropy encoding

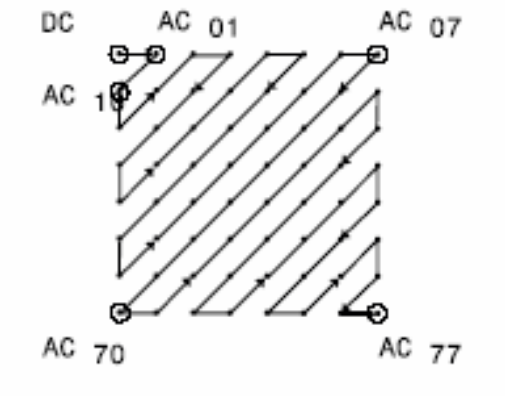


Figure 1-3

1.2.3. Entropy Encoding

The final step of the JPEG compression algorithm is the entropy encoding. There are two kinds of entropy encoding, the Huffman coding, and arithmetic coding. In this process the "zig-zag" quantized DCT coefficients are converted into intermediate symbols, and those symbols are converted into a data stream and a way that the symbols no longer have externally identifiable boundaries. In the case of Huffman coding, our case, there is a need for one or more sets of Huffman code sets that are provided by the application. Although arithmetic encoding produces 5-10 % better compression, its computational complexity makes it inappropriate for our case.

1.2.4. Color Transformation

The JPEG algorithm is capable of encoding images that use any type of color space. This is achieved because JPEG itself encodes each component in the color model separately, and is completely independent of any color-space model such as RGB, HIS or CMY. Yet the best results are obtained by input images in the luminance/chromance color space, such as YCbCr. The results are better because in the YCbCr color space most the visual information to which human eyes are most sensitive is found in the high-frequency gay-scale luminance component (Y) of the YCbCr color space. The other two components (Cb and Cr), contains color information to which the human eye is less sensitive and therefore they can be discarded.

In the case of color images, our case, the image contains from 1 to 256 image components (bands), each one been a rectangular shaped array of samples, with each sample consisting of unsigned integers with 8 bit precision.

In order to compress the color image, the color components must be downsampled are specific frequencies.

1.2.5. Downsampling color components

Although not specified in the JPEG standard, a downsampling of the color components must be address. When the uncompressed data is supplied in a conventional format, JPEG reduces the resolution of the chrominance channels by Downsampling or averaging together groups of pixels. The JPEG standard allows several different choices for the sampling frequencies of the downsampled channels.

Due to the large significance of the luminance component as discussed above, the luminance channel is left at full resolution i.e. 1:1 sampling. Generally, the two chrominance components are downsampled 2:1 horizontally and either 1:1 or 2:1 vertically. As a result the chromance pixel covers the same area as either a 2x1 or a 2x2 block of luminance pixels. JPEG refers to this process of downsampling as 2h1v and 2h2v, very often notated as 4:2:2 and 4:2:0 correspondingly. In 4:2:2, sampling the two chrominance pictures (Cb and Cr) process half the resolution in the horizontal direction and the full resolution in the vertical direction, compared to the luminance (Y) channel. In 4:2:0, both chrominance channels process half their resolution in both directions.

1.2.6. Encoding Order and Interleaving

A practical image compression standard must address how systems will need to handle the data during the process of decompression. Many applications need to

Pipeline the process of displaying or printing multiple-component images in parallel with the process of decompression. For many systems, this is only feasible if the components are interleaved together within the compressed data stream. To make the same interleaving machinery applicable to both DCT-based and predictive codecs, the JPEG proposal has defined the concept of "data unit." A data unit is a sample in predictive codecs and an 8x8 block of samples in DCT-based codecs. The order in which compressed data units are placed in the compressed data stream is a generalization of raster-scan order. Generally, data units are ordered from left-to-right and top-to-bottom. (It is the responsibility of applications to define which edges of a source image are top, bottom, left and right.) If an image component is no-interleaved (i.e., compressed without being interleaved with other components), compressed data units are ordered in a pure raster scan.

When two or more components are interleaved, each component C_i is partitioned into rectangular regions of H_i by V_i data units, as shown in the generalized example of Figure 8. Regions are ordered within a component from left-to-right and top-to-bottom, and within a region, data units are ordered from left-to-right and top-to-bottom. The JPEG proposal defines the term Minimum Coded Unit (MCU) to be the smallest group of interleaved data units. For the example shown, MCU_1 consists of data units taken first from the top-left-most region of C_1, followed by data units from the same region of C_2, and likewise for C_3 and C_4. MCU_2 continues the pattern as

shown. Thus, interleaved data is an ordered sequence of MCUs, and the number of data units contained in an MCU is determined by the number of components interleaved and their relative sampling factors. The maximum number of components which can be interleaved is 4 and the maximum number of data units in an MCU is 10. The latter restriction is expressed as shown in Equation 6, where the summation is over the interleaved components:

$$\sum H_i \times V_i \leq 10 \qquad \forall i \text{ in interleave}$$

Because of this restriction, not every combination of 4 components which can be represented in noninterleaved order within a JPEG-compressed image is allowed to be interleaved. Also, note that the JPEG proposal allows some components to be interleaved and some to be noninterleaved within the same compressed image.

1.3. Decompression

The decompression is the symmetric algorithm of the encoding. Figure 1-4 illustrates the step to decompress a JPEG image

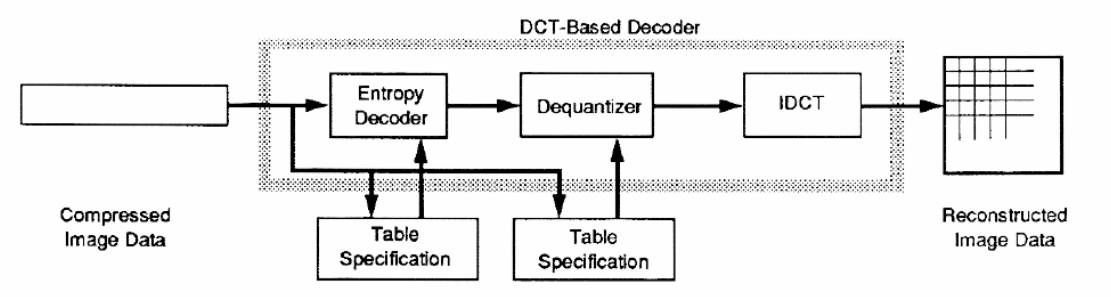


Figure 1-4

Where the IDCT in performed according to eq. 1

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16} \qquad \text{eq. 1}$$

Where: $C(u), C(v) = 1/\sqrt{2}$ for u,v=0

1 otherwise

1.4. Baseline Encoding Example

This section gives an example of Baseline compression and encoding of a single 8x8 sample block. Note that a good deal of the operation of a complete JPEG Baseline encoder is omitted here, including creation of Interchange Format information (parameters, headers, quantization and Huffman tables), byte-stuffing, padding to byte-boundaries prior to a marker code, and other key operations. Nonetheless, this example should help to make concrete much of the foregoing explanation.

Figure 1-5(a) is an 8x8 block of 8-bit samples.. The small variations from sample to sample indicate the predominance of low spatial frequencies. After subtracting 128 from each sample for the required level-shift, the 8x8 block is input to the FDCT. Figure 1-5(b) shows (to one decimal place) the resulting DCT coefficients. Except for a few of the lowest frequency coefficients, the amplitudes are quite small.

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

(a) source image samples

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

(b) forward DCT coefficients

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(c) quantization table

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(d) normalized quantized coefficients

240	0	-10	0	0	0	0	0
-24	-12	0	0	0	0	0	0
-14	-13	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(e) denormalized quantized coefficients

144	146	149	152	154	156	156	156
148	150	152	154	156	156	156	156
155	156	157	158	158	157	156	155
160	161	161	162	161	159	157	155
163	163	164	163	162	160	158	156
163	164	164	164	162	160	158	157
160	161	162	162	162	161	159	158
158	159	161	161	162	161	159	158

(f) reconstructed image samples

Figure1-5

1.5. Image, frame, and scan

Compressed image data consists of only one image. An image contains only one frame in the cases of sequential and progressive coding processes; an image contains multiple frames for the hierarchical mode. A frame contains one or more scans.

Finally, a scan contains a complete encoding of one (non-interleaved) or more image components (interleaved).

Related to the concepts of multiple-component interleave is the *minimum coded unit* (MCU). If the compressed image data is non-interleaved, the MCU is defined to be one data unit (8x8 block for DCT-based processes). If the compressed data is interleaved, the MCU contains one or more data units from each component.

For example consider an image in Y, Cb, Cr format with 4:2:0 sampling as seen in Fig. 1.5. This image has three components. If the compressed data is interleaved, there would be only one scan, and the first and second MCU would be:

$$MCU_1 = Y_{00}Y_{01}Y_{10}Y_{11}Cb_{00}Cr_{00}$$
$$MCU_2 = Y_{02}Y_{03}Y_{12}Y_{13}Cb_{01}Cr_{01}$$

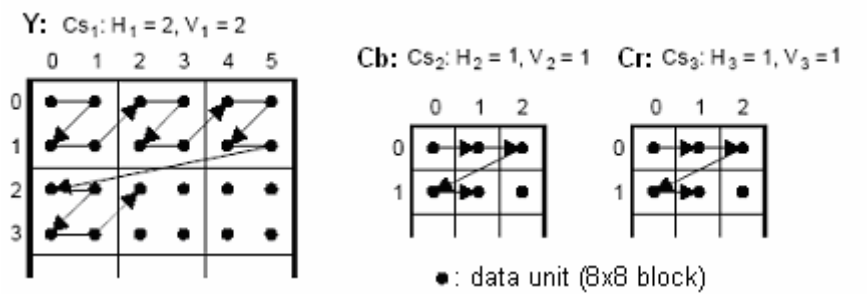


Figure 1-6: Interleaved Components

If the compressed data is non-interleaved, there would be three scans. The first scan would contain the coded luminance (Y), the second the coded Cb component and the last the Cr. For each scan the MCU is a data unit.

The MCU's in either case (interleaved, non-interleaved) are encode in raster scan order as seen in fig. 1-7.

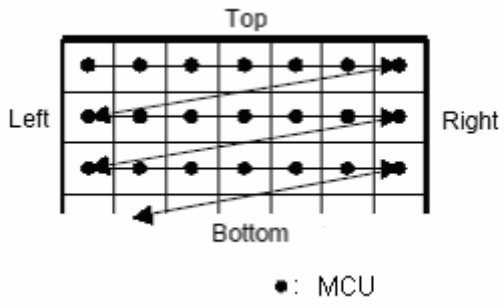


Figure 1-7: Coding order

1.6. The JFIF (JPEG File Interchange Format)

1.6.1. The JPEG Bitstream

The when the image is compressed it is transform into a bitstream.

Each compressed image can be divided into group. The informations are encoded as shown in figure 1-8.

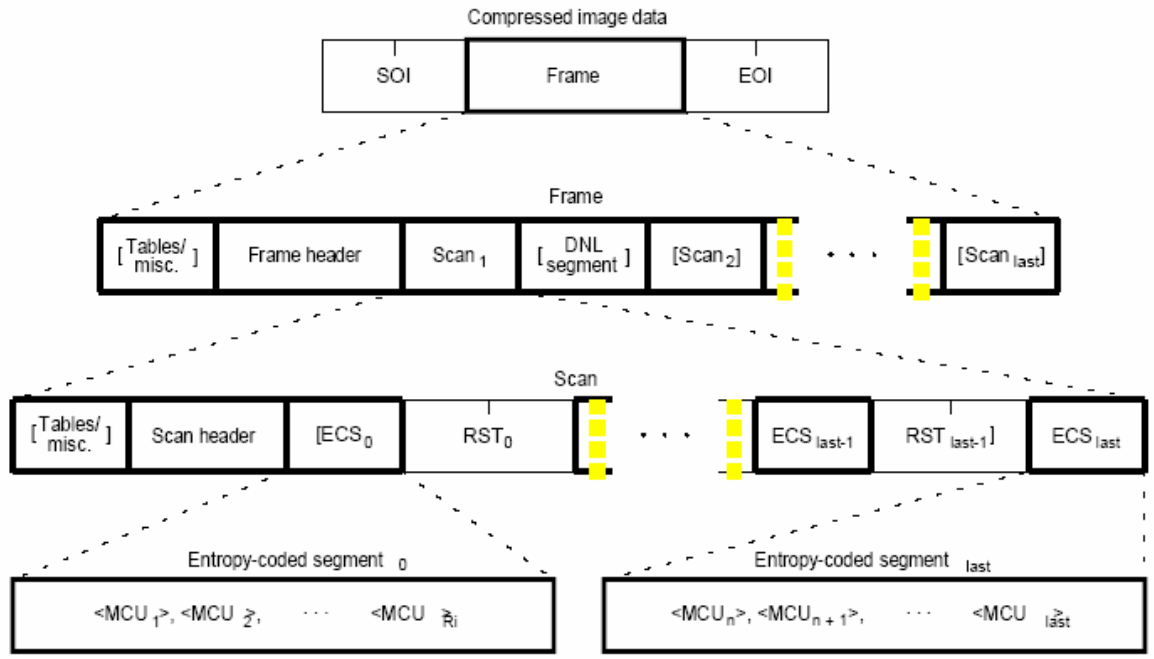


Figure 1-8

The markers shown in Figure 1.8 are defined as follows:

SOI: Start of image marker – Marks the start of a compressed image represented in the interchange format or abbreviated format.

EOI: End of image marker – Marks the end of a compressed image represented in the interchange format or abbreviated format.

RSTm: Restart marker – A conditional marker which is placed between entropy-coded segments only if restart is enabled. There are 8 unique restart markers (m = 0 - 7) which repeat in sequence from 0 to 7, starting with zero for each scan, to provide a modulo 8 restart interval count. The encoder outputs the restart markers, intermixed with the entropy-coded data, at regular *restart intervals* of the source image data.

Restart markers can be identified without having to decode the compressed data to find them. Because they can be independently decoded, they have application-specific uses, such as parallel encoding or decoding, isolation of data corruptions, and semi-random access of entropy-coded segments.

The top level of figure 1.8 specifies that the non-hierarchical interchange format shall begin with an SOI marker, shall contain one frame, and shall end with an EOI marker.

The second level of figure 1.8 specifies that a frame shall begin with a frame header and shall contain one or more scans. A frame header may be preceded by one or more table-specification or miscellaneous marker segments as described in the next section. In the table specification header are defined the quantization matrices. Up to 4 matrices may be specified, each used to quantize one component. For sequential DCT-based and lossless processes each scan shall contain from one to four image components. If two to four components are contained within a scan, they shall be interleaved within the scan.

The third level of figure 1.8 specifies that a scan shall begin with a scan header and shall contain one or more entropy coded data segments. Each scan header may be preceded by one or more table-specification or miscellaneous marker segments. If restart is not enabled, there shall be only one entropy-coded segment and no restart markers shall be present. If restart is enabled, the number of entropy-coded segments is defined by the size of the image and the defined restart interval. In this case, a restart marker shall follow each entropy-coded segment except the last one.

The fourth level of Figure 1.8 specifies that each entropy-coded segment is comprised of a sequence of entropy coded MCUs. If restart is enabled and the restart interval is defined to be Ri, each entropy-coded segment except the last one shall contain Ri MCUs. The last one shall contain whatever number of MCUs completes the scan.

The required table-specification data **must** be present at one or more of the allowed locations.

Table 1 contains all the markers present in the JPEG File Interchange Format

0xFFC0	SOF_0 - Baseline DCT
0xFFC1	SOF_1 - Extended sequential DCT
0xFFC4	DHT - Define Huffman Table(s)
0xFFD0	RST_0 - Restart with modulo 8 count 0
0xFFD1	RST_1 - Restart with modulo 8 count 1
0xFFD2	RST_2 - Restart with modulo 8 count 2
0xFFD3	RST_3 - Restart with modulo 8 count 3
0xFFD4	RST_4 - Restart with modulo 8 count 4
0xFFD5	RST_5 - Restart with modulo 8 count 5
0xFFD6	RST_6 - Restart with modulo 8 count 6
0xFFD7	RST_7 - Restart with modulo 8 count 7
0xFFD8	SOI - Start of Image
0xFFD9	EOI - End of Image
0xFFDA	SOS - Start of Scan
0xFFDB	DQT - Define Quantization Table(s)
0xFFDD	DRI - Define Restart Interval
0xFFE6	APP_6 - NITF application segment
0xFFFE	COM - Comment

Table 1

1.6.2. Frame header

The frame header which shall be present at the start of a frame specifies the source image characteristics (sample precision, dimensions), the components in the frame, and the sampling factors for each component, and specifies the destinations (see table specification syntax) from which the quantized tables to be used with each component are retrieved.

1.6.3. Scan header

The scan header which shall be present at the start of a scan specifies which component(s) are contained in the scan, specifies the destinations from which the entropy tables to be used with each component are retrieved.

If there is only one image component present in a scan, that component is, by definition, non-interleaved. If there is more than one image component present in a scan, the components present are, by definition, interleaved.

1.6.4. Table-specification and miscellaneous marker segment syntax

At the places indicated in Figure 1.9, any of the table-specification segments or miscellaneous marker segments specified may be present in any order and with no limit on the number of segments.

If any table specification for a particular destination occurs in the compressed image data, it shall replace any previous table specified for this destination, and shall be used whenever this destination is specified in the remaining scans in the frame or subsequent images represented in the abbreviated format for compressed image data. If a table specification for a given destination occurs more than once in the compressed image data, each specification shall replace the previous specification.

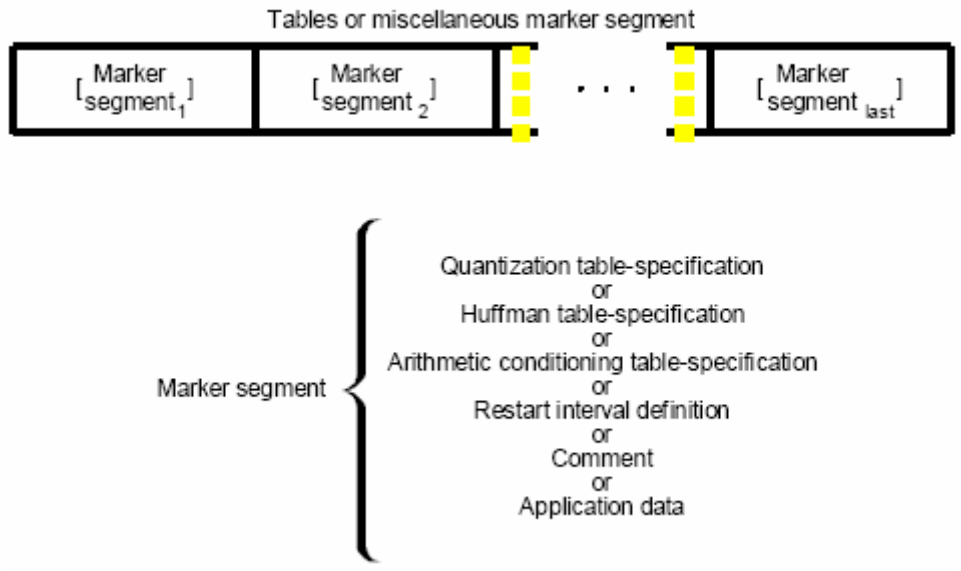


Figure 1-9: Table Specification or misc. Marker Segment

- The Huffman table-specification segment defines a Huffman table to be used for entropy coding.

- The Arithmetic conditioning table-specification replaces the default arithmetic coding conditioning tables established by the SOI marker for arithmetic coding processes.
- The Restart interval definition defines the restart interval.
- The Comment segment allows for user comments.
- The Application data are reserved for application use. Since these segments may be defined differently for different applications, they should be removed when the data are exchanged between application environments.

The quantization table-specification segment defines the quantization matrices and is described analytically in the next section.

1.6.5. Quantization table-specification syntax

Figure 1.10 specifies the marker segment which defines one or more quantization tables.

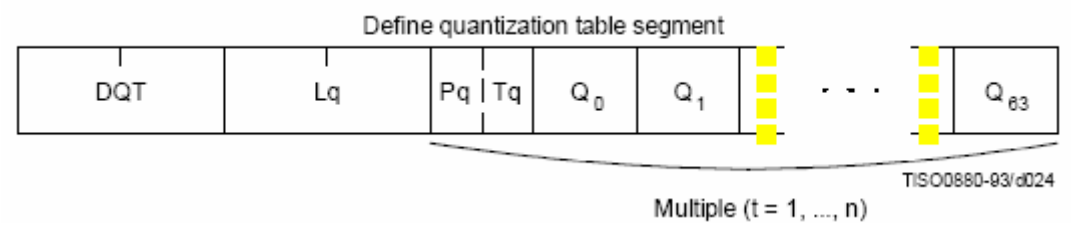


Figure 1-10: Quantization table Specification

The marker and parameters shown in Figure 1.10 are defined below.

DQT: Define quantization table marker – Marks the beginning of quantization table-specification parameters.

Lq: Quantization table definition length – Specifies the length of all quantization table parameters shown in Figure 1.10.

Pq: Quantization table element precision – Specifies the precision of the Qk values. Value 0 indicates 8-bit Qk values; value 1 indicates 16-bit Qk values. Pq shall be zero for 8 bit sample precision P.

Tq: Quantization table destination identifier – Specifies one of four possible destinations at the decoder into which the quantization table shall be installed.

Qk: Quantization table element – Specifies the *k*th element out of 64 elements, where *k* is the index in the zigzag ordering of the DCT coefficients. The quantization elements shall be specified in zig-zag scan order.

The value n in figure 1.10 is the number of quantization tables specified in the DQT marker segment.

Once a quantization table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used, when referenced, in the remaining scans of the current image and in subsequent images represented in the abbreviated format for compressed image data. If a table has never been defined for a particular destination, then when this destination is specified in a frame header, the results are unpredictable.

An 8-bit DCT-based process shall not use a 16-bit precision quantization table.

1.6.6. Application Data

Figure 1.11 specifies the marker segment structure for an application data segment

Figure 1-11: Application Data Segment

APPn: Application data marker – Marks the beginning of an application data segment.

Lp: Application data segment length – Specifies the length of the application data segment.

Api: Application data byte – The interpretation is left to the application.

The APPn (Application) segments are reserved for application use. Since these segments may be defined differently for different applications, they should be removed when the data are exchanged between application environments.

CHAPTER 2

Network Protocol Overview

2.1. Protocols

Due to the huge amount of data that we need to transmit the network protocols must be wisely selected. According to the Open Systems Interconnection (OSI) networking suite the network protocols are grouped into the seven layer OSI model. Those are the Application, Presentation, Session, Transport, Network, Data link and Physical layer, grouped as shown in figure 2.1

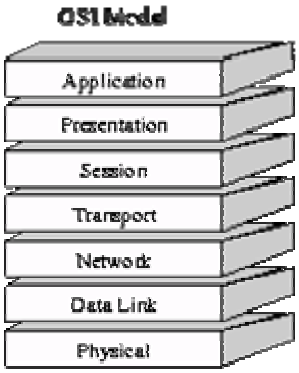


Figure 2-1

Commonly, the top three layers of the OSI model (Application, Presentation and Session) are considered as a single Application Layer in the TCP/UDP/IP suite. Because the TCP/UDP/IP suite has no unified session layer on which higher layers are built, these functions are typically carried out (or ignored) by individual applications. The most notable difference between TCP/UDP/IP and OSI models is the Application layer, as TCP/IP integrates a few steps of the OSI model into its Application layer.

Therefore the simplified case, is shown in figure 2.2

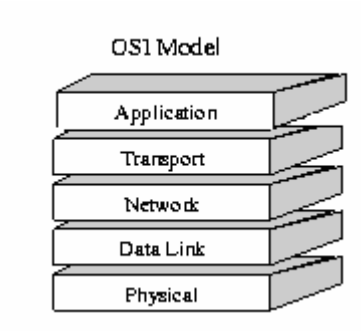


Figure 2-2

In the Application layer resides our application. In view of the application layer demands, our application is responsible for placing a JPEG image into a number of packets and transmitting them form the server side, and reconstruction the JPEG image form the packet, in the client side. A more detailed explanation of the way the JPEG image is cut into packets and how those packets are regrouped to create the image is given latter.

In the Transport layer, there are two main protocol form which to choose. The User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). Each one is explained later and a comparison between them is performed. We final choose UDP for the data channel and TCP for the control channel.

In the network layer, the IP is the only available protocol that suites are needs, that is IP is used by source and destination to communicate packets, over a pocket-switched network, like internet. The other protocols at the network layer are mainly for routing, which is out of the scope of our case.

The Data Link layer and the Physical layer are mainly for describing the very low level of the communication. The Data Link layer protocol is the Ethernet and the Physical layer describes the physical media between the sender and the receiver.

2.2. Application Programming Interface

In order to control UDP or TCP, we use a programming interface (API) called Berkley Socket, or just socket. Sockets are a C programming language library for UNIX, but nowadays it is available at any operating system, e.g. Linux, Windows etc. There are two main functions for control the UDP, through a socket, send and receive.

In the Windows environment both send and receive functions get the following attributes:

S [in] Descriptor identifying a connected socket.

Buf [in] Buffer containing the data to be transmitted.

Len [in] Length of the data in *buf*, in bytes.

Flags [in] Indicator specifying the way in which the call is made.

2.2.1. TCP

The Transmission Control Protocol (TCP) is the most widely use Transport protocol. The protocol guarantees that the data to be send are going to be received intact, in the same order as they were send and that they will be directed to the correct application (Multiplexing – Demultiplexing). The protocol is documented in the IETF RFC 793.
The TCP has an 224 bytes header, containing information such as Source Port Destination Port, Sequence Number, Acknowledgement Number, Checksum and others. Applications send streams of 8-bit bytes to TCP for delivery through the network, and TCP divides the byte stream into segments, of size defined into to the maximum transmission unit (MTU) and passess them to the data link layer of the network the computer is attached to . TCP then passes the resulting packets to the Internet Protocol, for delivery through an internet to the TCP module of other host . TCP checks to make sure that no packets are lost by giving each byte a sequence number, which is also used to make sure that the data are delivered to the other host in the correct order. The TCP module at the other host sends back an acknowledgement for packets which have been successfully received; a timer at the sending TCP will cause a timeout if an acknowledgement is not received within a reasonable round-trip time (or RTT), and the (presumably lost) data will then be re-transmitted. The TCP checks that no bytes are damaged by using a checksum; one is computed at the sender for each block of data before it is sent, and checked at the receiver.

2.2.2. UDP

The User Datagram Protocol (UDP) on the other hand does no guarantees the delivery of the packets the sender has send, nor does is guarantees that the will be received at the correct order that they are send. It does though guarantee that packets will be directed to the correct application (Multiplexing – Demultiplexing). The UDP header is only 64 bytes long, containing the Source Port, Destination Port, Length and Checksum.

2.2.3. TCP versus UDP for video transmission

Comparing TCP and UDP, TCP offers a reliable delivery and good congestion management. Nevertheless, the need of acknowledgements and the inability of process out-of-order packets make it ill-suited for our case. That is the reason why we choose UDP as a transport layer protocol. UDP takes a packet of data, adds an

appropriate header mainly specifying the source and the destination and sends it to the network layer. Because it does not care whether the packet has reached its destination of not, any congestion management it left to the application. Therefore our application must be designed so as to overcome the lack of congestion management and must be able to process out-of-order packets.

Although the UDP has clear advantages over the TCP as far as the transmission of the video is concerned. Nevertheless TCP protocol is used in the control connection established between the server and the client. The reason is that information passing through the control channel is critical and therefore must reach their destination and because the bandwidth needed for this communication is very small comparing to bandwidth needed for transmitting the video.

The use of two distinct channels, one for control and the other for data, between the server and each client, makes the protocol out-of-bound.

CHAPTER 3

The Implementation

Implementation Sub-Systems

3.1. The Server

3.1.1. Initialize the Server

Initially the server binds to a local port in order to receive requests from the clients. The binding is performed by creating a TCP socket in this port. The server creates a thread that is listening for incoming requests. A soon as the server receives a request, it create a UDP socket which is bind to one of his local ports and starts transmitting the video through this socket to the client. The information needed in order to locate and send the video to the client, is the client's IP address and local port. This information is received in the initial message send to the server by the client. While transmitting the video to the client, the server keeps listening for incoming connection. At a given time, the server can communicate with up to 10 clients, although that figure is easy modified to support up to around 60000 different clients.

3.1.2. The image acquisition

The server is responsible from sending the video just captured. At a given point a new image is grabbed. Using an image processing library the image is processed and compressed. The image is compressed using the JPEG lossy compression algorithm with the restart markers option enabled and set to a specific value.

3.1.3. Datagrams

The communication of the video is performed using UDP sockets. A UDP socket can transmit only datagrams, it does not directly transmit image. Therefore it is necessary to place the image into datagrams. The maximum size of a UDP datagram is about 8Kbytes which is far small to contain a full image, especial in our case, where image are typically about 150 Kbytes. This implies that the image must be partitioned to a number of datagram. Its datagram is placed into a packet at the network layer. So, in fact we must partition the image in a certain way, in order to be able to reconstruct the

image from the various packets. Moreover, the packets may arrive out of order, which implies that care must be taken to support the re-ordering of the packets as they arrive. In order to facilitate these requirements, the image is partitioned according to the restart markers located in it.

3.1.4. Creating the datagram

The restart markers are part of a number of markers, as discussed above. The location of the markers is performed in a serial way. For every marker located a number of informations are stored into a data structure. Those information are the absolute position in the image (in bytes), the number of bytes until the next marker, indicating the size of the segment and the type of the marker i.e. the first restart marker, the fifth restart marker, the start of frame marker etc.

3.1.5. Packing the datagram

Using this data structure, we start placing image segment until the packet is filled, or until the END-OF-IMAGE marker is found, indicating the end of the image. In latter, case the header of that packet is modified to specify the end of the image. Each packet can contain an integer number of segments, therefore, each packet must start with a Restart Marker. For every packet, we start from a point in the image, indicated by a variable, and stop as soon as one of the two constraints discussed above is meet. At that time, we store the relative position inside the image and the corresponding Restart Marker. The next packet is filled from the next segment until one of the constraints set above is satisfied. The packaging process is illustrated in figure 3.1

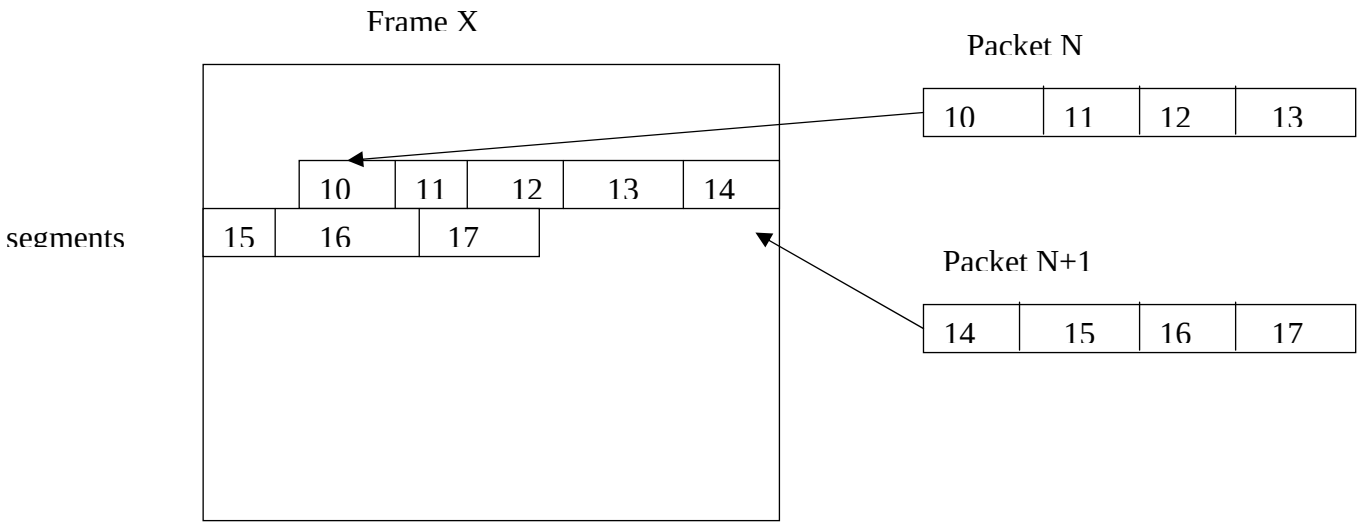


Figure 3-1

3.1.6. Creating the header

In order to support out-of-order delivery, a header with information about the packet received must be attached to the packet. The header is created after filling the packet with image data. The header carries information in order for the client to understand the relation of the packet to the full image and support the reconstruction of the image in case of a packet loss. Those informations are:

I. The frame number, necessary for the client in order to avoid placing the packet in another frame. Since the UDP protocol cannot guarantee the correct order of delivery, the application must be able to distinguish the frames.

II. The accumulative number of the first segment the packet contains and the number of segments the packet contains. A packet can contain many segments. Therefore, when a packet is lost, a number of segments are lost. In order to reconstruct the frame, we need to know what segments are missing. Although we cannot directly access the segments, we bypass this problem by using the Restart Markers. Each Restart Marker signifies the begging of a number of segments. Knowing the first Restart Marker and the number of Restart Markers, we can tell what the missing segments are.

III. The offset from the start of the image, where we start copying the segments to the packet. This information is used by the client in order to copy the received

packet at exactly the same location in the buffer as it was copied from the frame.

When the end of the image is reached, the header contains the End-of-Image marker and the number of packets the server has send for this frame. The client can uses the number of packets to verify the number of missing packets.

3.2. The Client

3.2.1. Initialize the client

Initially the client creates a UDP socket that is bound to a local port. The client initiates the connection by sending a packet to the server stating his IP address and the local port to which he is waiting to receive the video. If the server cannot be reached by the client, the clients keeps trying to connect, in case there is excessive traffic to the network or the server is temporary unavailable. As soon as the server is reached, it starts transmitting the video to the client, at the IP address and the port specified in the request message.

3.2.2. Receive the packet

The client receives packet through the socket he specified. The packet are received out-of-order, therefore a re-ordering is performed. The packet is placed in the image buffer, according to the informations in the packet's header. Along with other information, the header indicates the absolute position (in bytes) of the first block that was placed in the packet. Since the rest of the blocks in the packet are placed sequentially, we start placing the image data in the packet at the same absolute position in the client's image buffer.

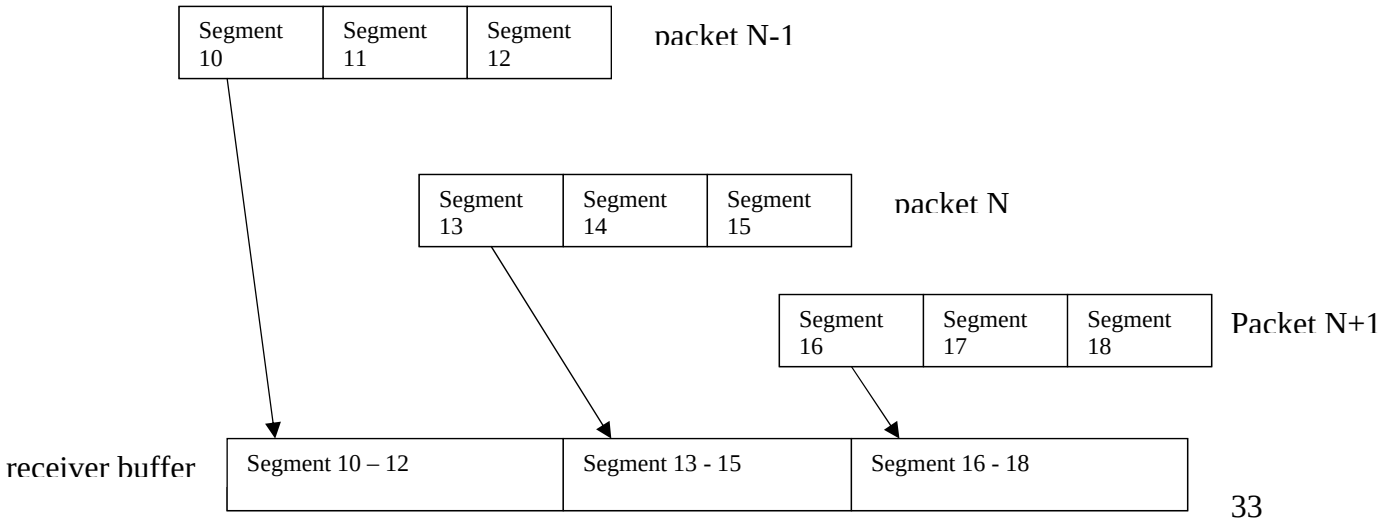


Figure 3-2

3.2.3. Check for errors

After receiving a packet, it is necessary to check if the packet is the expected one or not. This is check, by comparing the last packet's number with the packet number of the newly received packet. If the packet's number is the subsequent number of the previously received packet, then no error has occurred. We assume that the packet receive in error-free, since the error detecting code is define in the network lay (IP layer). Ideally, by the end of the transmission of the image, the image buffer in the server must be identical with the image buffer in the client. If, on the other hand, the newly received packet' number incorrect, we identify the packets missing and store this information into a data structure. There are two types of error in the received packet, either the packet's number is bigger than expected or it is smaller. In the former case we assume that the packets between the last packet received and the newly received have been lost. In the other case, we assume that the rest of the last image packet has been lost, and so have the first packet of the image represented by this packet. In either case, a special error code is returned indicating the type of error detected.

3.3. The Error Concealment Techniques

When a newly received image raises such an error code, the reconstruction function takes over. When a packet is lost during the transport from the client to the server, a portion of the image is lost. Due to the entropy encoding of the image, a missing part of the bitstream can result in total destruction of the image. There are three kinds of concealment algorithm; the spatial, the temporal and the hybrid.

3.3.1. Spatial redundancy based concealment algorithms

The spatial error concealment algorithms rely on the inherited spatial smoothness found the any image. The main idea is to conceal the error using neighbouring available pixel coefficients. Although this technique is very simple, is does not has good results. This is due to the blurring of the edges in the image, and the general inability to reconstruct complex patterns. With some extra effort, much better results may be presented by integrating an edge-based directional interpolation. A very

simple scheme has been also proposed in [8], where using the pixel-wise difference between two sets of projection data available on the block boundary, a better estimation of the edge direction is performed. Projection onto convex sets (POCS) has been proposed, which iteratively use the spatial redundancy and the pixel value constraint. The performance of such a scheme is depended on the accuracy of the estimation about spatial redundancy and pixel values.

3.3.2. Temporal redundancy based concealment algorithms

Temporal error concealment, exploit the temporal redundancy inherited in a video. They use temporally adjacent frames to evaluate the missing information. The idea is based on the fact that video content is smooth and continuous in the time domain. One scheme is to just copy the missing blocks from the previous frame. This assumes that the motion vector (MV) is near zero. A near-zero motion vector implies a little or no motion in the scene. This is the typical case for surveillance systems. A different approach is to use the motion vectors from the previous image. In scenes with regular motion, this approach has good result. Unfortunately, due to the real-time nature of our algorithm, the extra processing power used in evaluating the motion vector, may result is a drop of the frame rate. Thus, this approach is inefficient is our technique.

3.3.3. Hybrid spatial-temporal based concealment algorithms

Some effort has been made in order to combine both spatial and temporal concealment algorithms, in order to improve there performance. An example is the use of smoothing constraint applied to both spatial and temporal, in order to produce a more smooth concealed block. The problem with this approach is the possible loss of high frequency components. Although more recent techniques have been developed, in general there has been little success in joint effort. This is because there is lack in efficient ways of merging the results in both spatial and temporal error concealment. The extra complexity involved in such a process is also of significant effect, again due to the limited time and processing power available in such types of applications.

3.4. The Error Concealment Algorithm Implementation

In our implementation, we have used only the temporal based error concealment technique. The choice was made upon examining the video in question and the real-time constraint set on our model. Only one technique was chosen to be implemented, due to the relatively low increase in image quality of the hybrid spatial-temporal techniques and the increase of the computational complexity of implementing both spatial and temporal concealment technique. Furthermore, our implementation's main use in surveillance, where we usually have little movement and therefore little difference in successive frames. The key feature used in implementing our concealment technique was the use of Restart Markers. Restart Markers are spatial code words that are used in the encoding of the JPEG frames, that make up the M-JPEG video.

3.4.1. The use of Restart Markers

Each packet can contain a number of segments. These segments are distinguished using Restart Markers. Each segment encodes a predefined number of 8 x 8 blocks. Since the images have fixed dimensions, both width and length, the number of blocks that make the image is also fixed. Therefore, the number of Restart Markers is also fixed. For Example,

an 800 x 800 image has 100*100 =10 000 blocks and 10 000 – 1 Restart Markers. Although the number of blocks, segments and Restart Markers are fixed, none of them have a constant size in terms of bytes. This variation is a consequence of the entropy encoding stage in the JPEG compression algorithm. Since most of the AC coefficients produced by the DCT encoding are zero or near-zero, an 8 x 8 block can vary form 1 (only the DC coefficient) up to 64 coefficients (the DC and all the AC coefficients). As a result a segment, which is made up form a fixed number of blocks, can have variable size. Since a packet is made up form segments, the variation of the segment's size implies a variation in the number of segments that a fixed size packet can carry. For example, a 4Kbyte packet can contain from 10 large size segments to 200 small size segments.

In terms of error concealment, the variation of segments contained in a packet, makes a direct copy from the corresponding packet received in the previous image to the place normally occupied from the missed packet, ineffective. This inefficiency is due to the variable number of segments contained in the previously received packet and the variation of the missing packet's number of segments. The above are illustrated in figure 3.3.

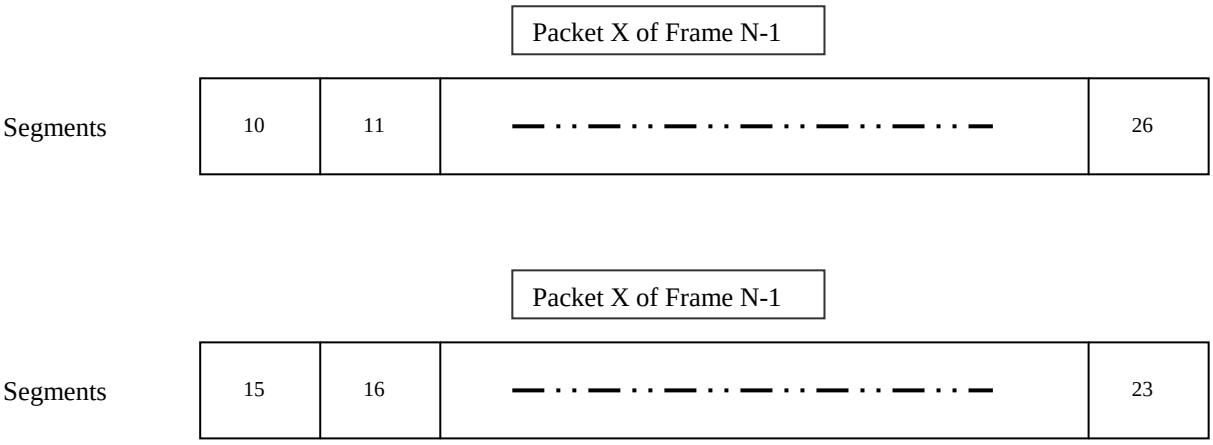


Figure 3-3

In order to overcome the problem, we use the Restart Markers is an accumulative way. Each packet's header carries the information of the first accumulative Restart Marker and the number of Restart Markers contained in the packet. The number of the Restart Markers contained in the packet is the same as the number of the segments contained in the image. Since the segments are evaluated in the spatial domain, the nth segment is corresponding to the blocks located at specific position in the image and they are the same for every image transmitted, as shown in figure 3.4

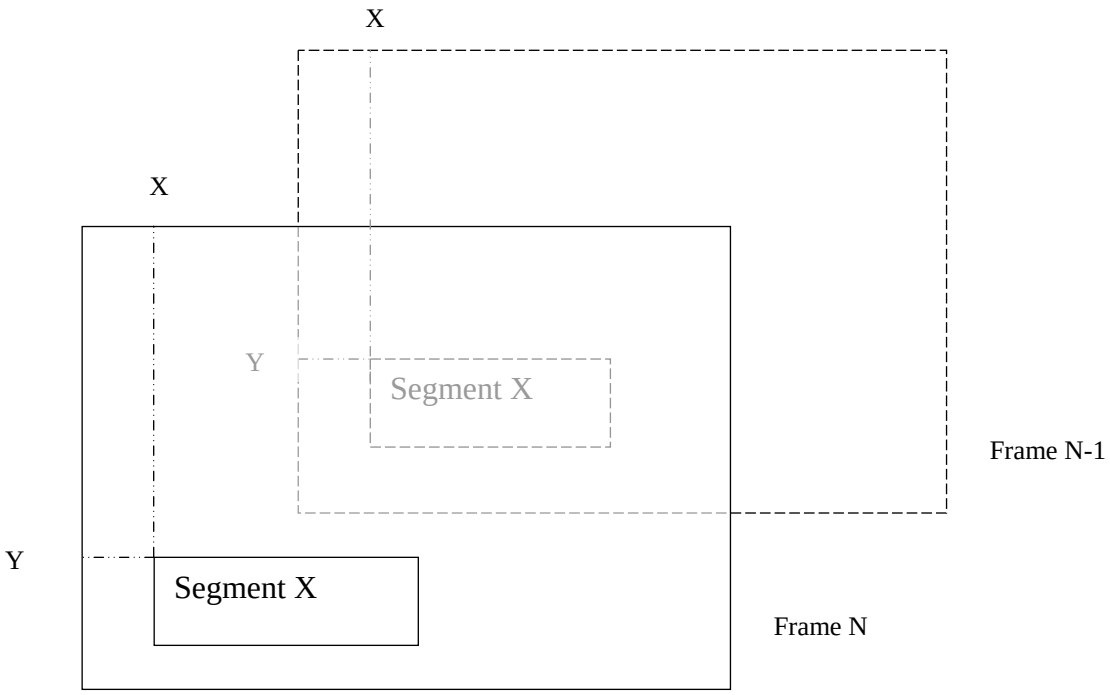
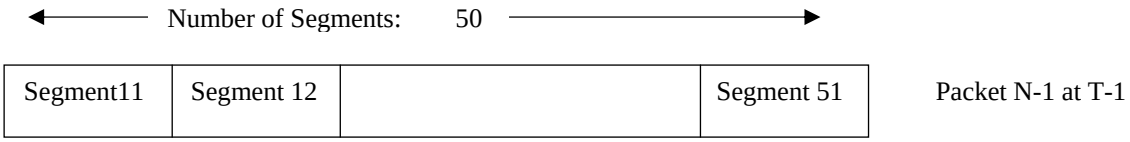


Figure 3-4

In order to reconstruct the image all the Restart Markers of the frame must be located. First, we must find the missing segment. This is done by examining the packet that was received just before a packet was missed. From the header of that packet we can find the first segment that was encoding in the missing packet, by adding the number of segments in the previous packet with the first segment located in that packet.

Then, we must find the number of segment that this packet was carrying. This can also be found by examining the header of the packet that just arrived. From this packet's header we can evaluate the number of missing segment by subtracting the first segment in the missing packet from the first segment encoded in the packet just arrived.



3.4.3. Creating the reconstructed image

Having located the first missing segment and the number of missing segment, we can reconstruct the frame based on previously received packets. When a packet is missing we search the packets received from the previous frame and we locate the missing segment in the previous frame. By knowing the first missing segment and the segments missing, we just copy the corresponding segment form the previous packets. For example, if the missed packet had segments form A to B, we copy the segments A to be from the previous frame. This process is illustrated in figure 3.6.

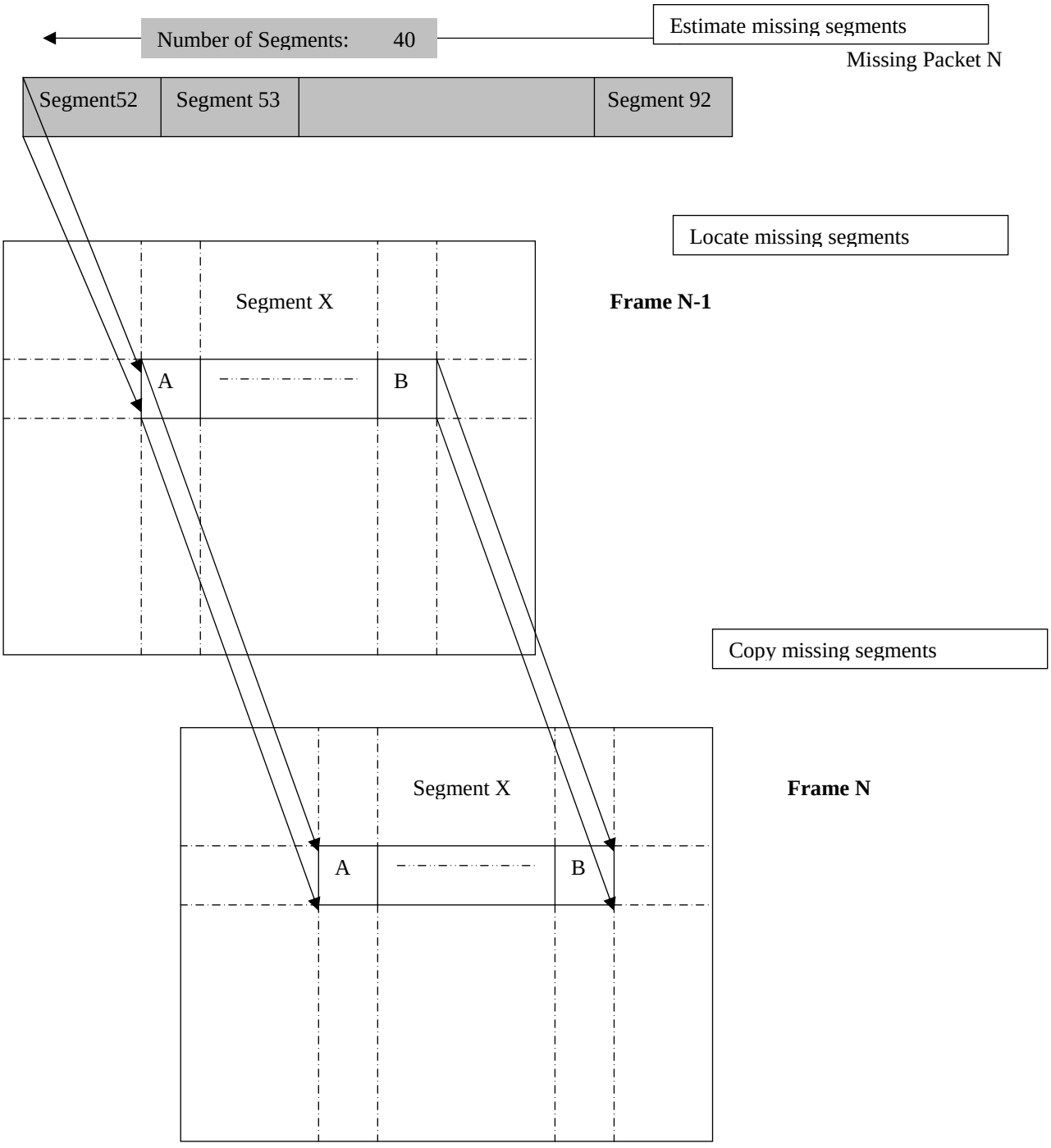

Number of Segments: 40
Estimate missing segments
Missing Packet N
Segment52
Segment 53
Segment 92
Locate missing segments
Segment X
Frame N-1
A
B
Copy missing segments
Segment X
Frame N
A
B

Figure 3-6

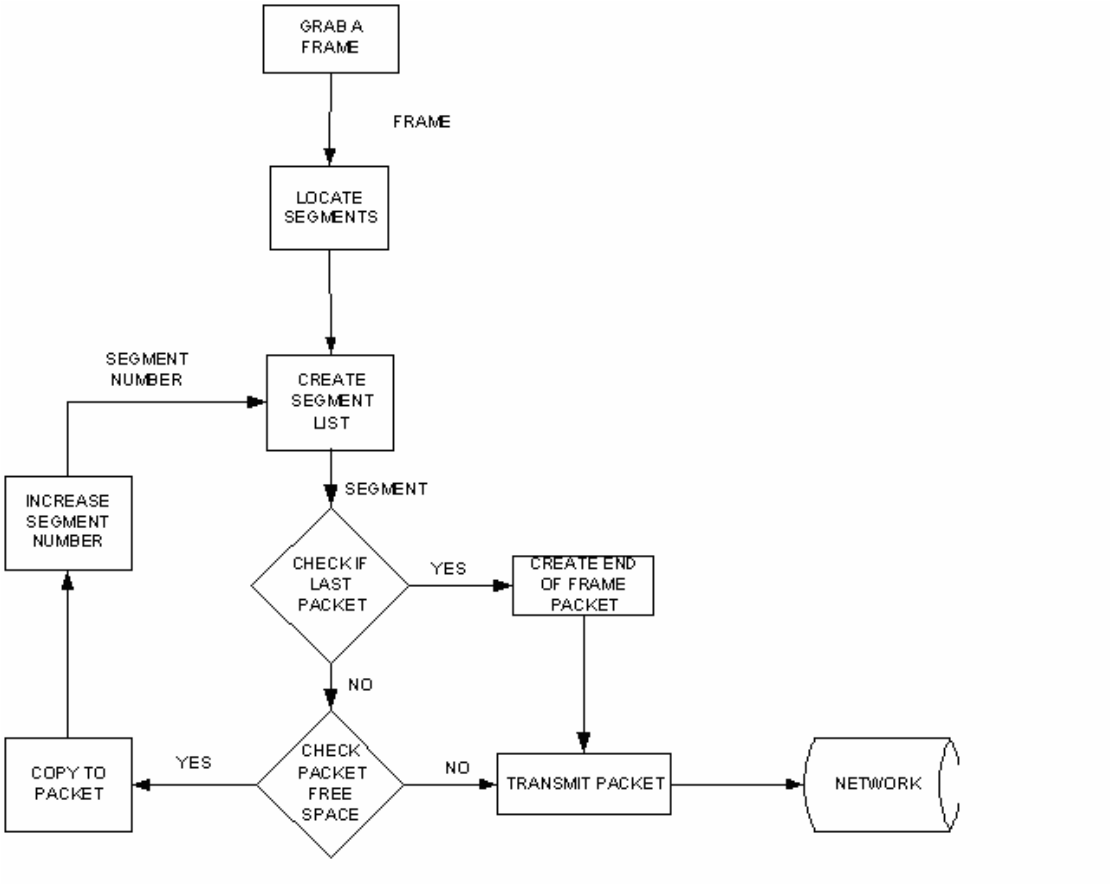


Figure 3-7: Block Diagram of the Server

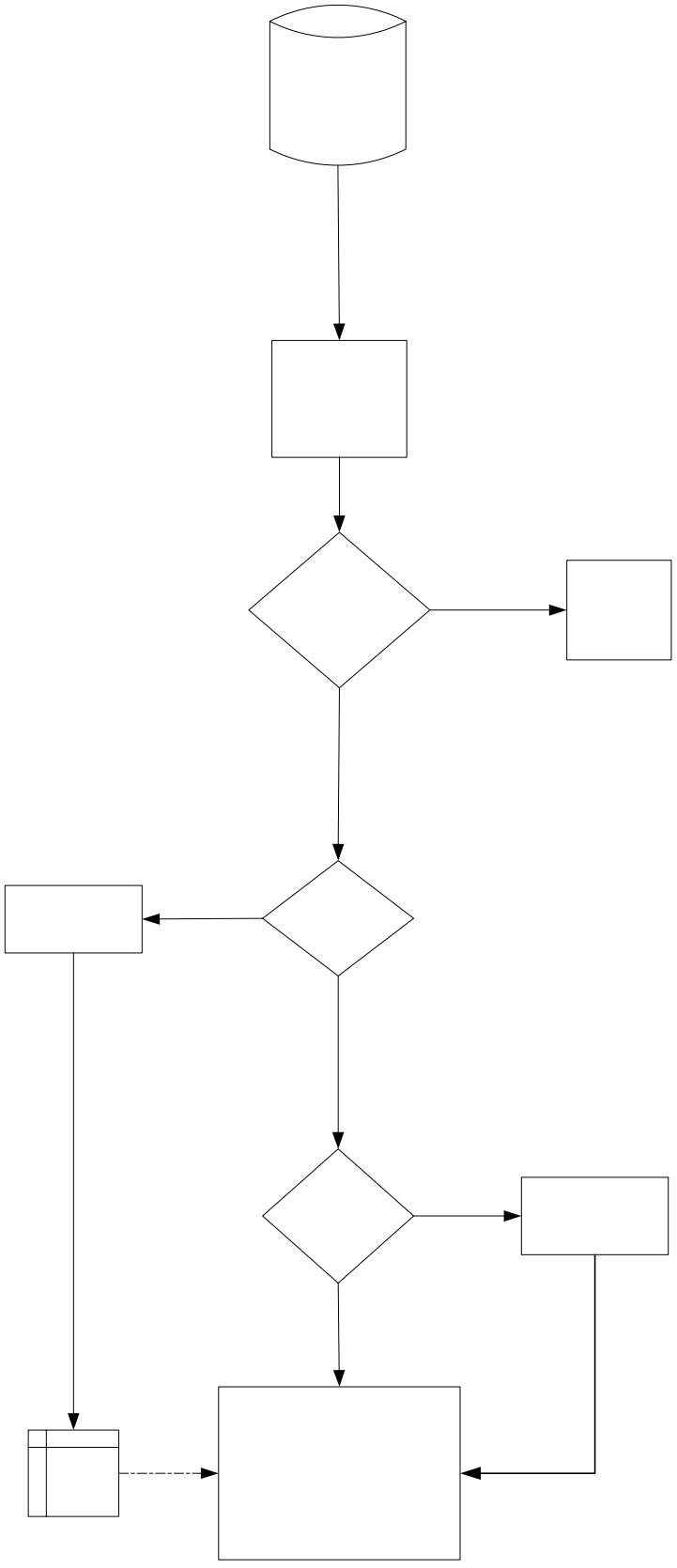


Figure 3-8: Block Diagram of the Client

CHAPTER 4

Experimental Results

4.1. Theoretical Background

The experimental results are separated into two groups. In the first group we measured the quality of the received video when there is no motion in the transmitted video. In the second group we measured the quality of the received video when there is some motion. The motion is measured in terms of frame-to-frame pixel-wise difference. Although, the frame-to-frame pixel-wise difference is not a good metric of the motion inside the video, it give a good hind as to weather two successive frames are similar enough to produce good results in terms of our error-concealment algorithm. The relation of the error-concealment algorithm and the motion of the video in very close, because of the temporal redundancy of the video. The more motion there is in the video stream, the smaller the temporal redundancy, and the poorer the missing packet estimation is.

In order to measure the quality of the received video, we calculate the Mean Square Error (MSE) and the Peak Signal to Noise Ration (PSNR) in decibels (db),
using eq 4.1

For a N by N pixels Image

$$MSE = \frac{\sum_{X=0}^{N} \sum_{y=0}^{N} \left[f(x,y) - F(x,y) \right]^2}{N^2}$$

$$PSNR = 20 * \log_{10} \left(\frac{255}{RMSE} \right) \text{ Where } \quad RMSE = \sqrt{MSE}$$

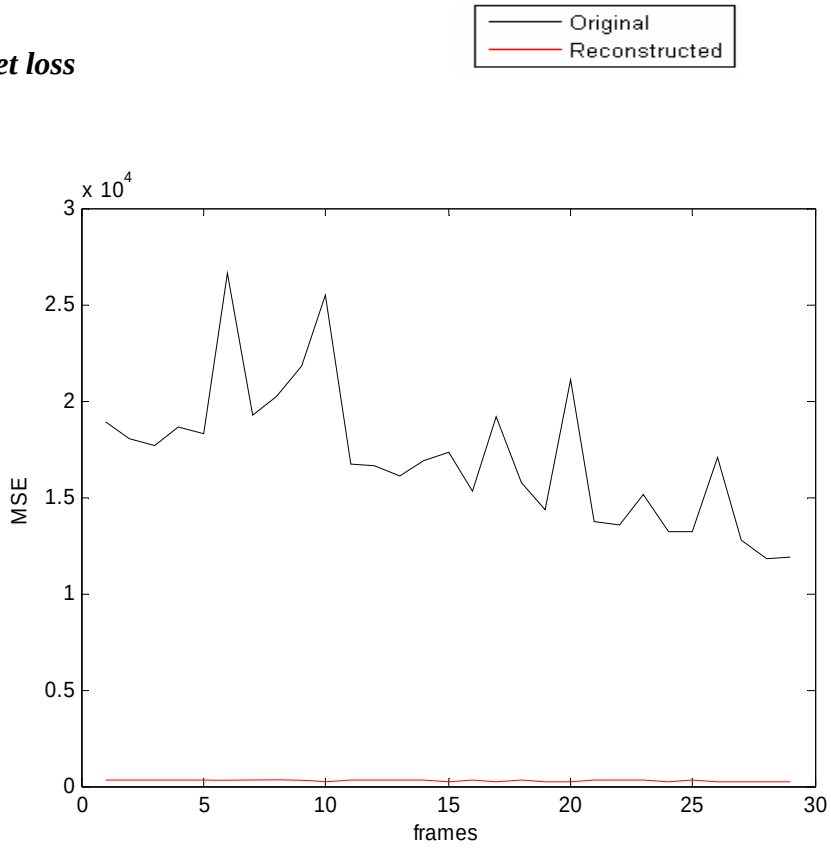
eq .4.1

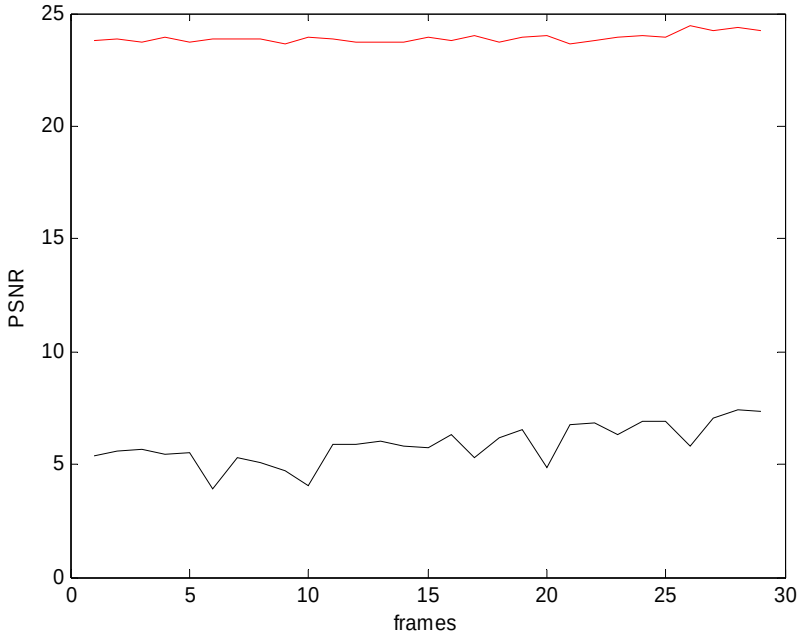
For each of the two group we measure the quality of the received image when 3,6 and 10 % of the packets consisting the image are lost

In the experiments we consider the case of 3, 6 and 10 % packet loss, while we do not consider the available bandwidth. The idea is that in order to transmit the necessary information with no error, a minimum bandwidth for the channel must be set according to Shannon's theorem [10]. Assuming that the minimum bandwidth necessary is available in a mean value manner, we consider the cases where the deviations on the available bandwidth caused by the packet-switched network functions led to a specific bit error rate. For those bit error rate, we examine the effect on the received images and the quality of the error-concealment algorithm.

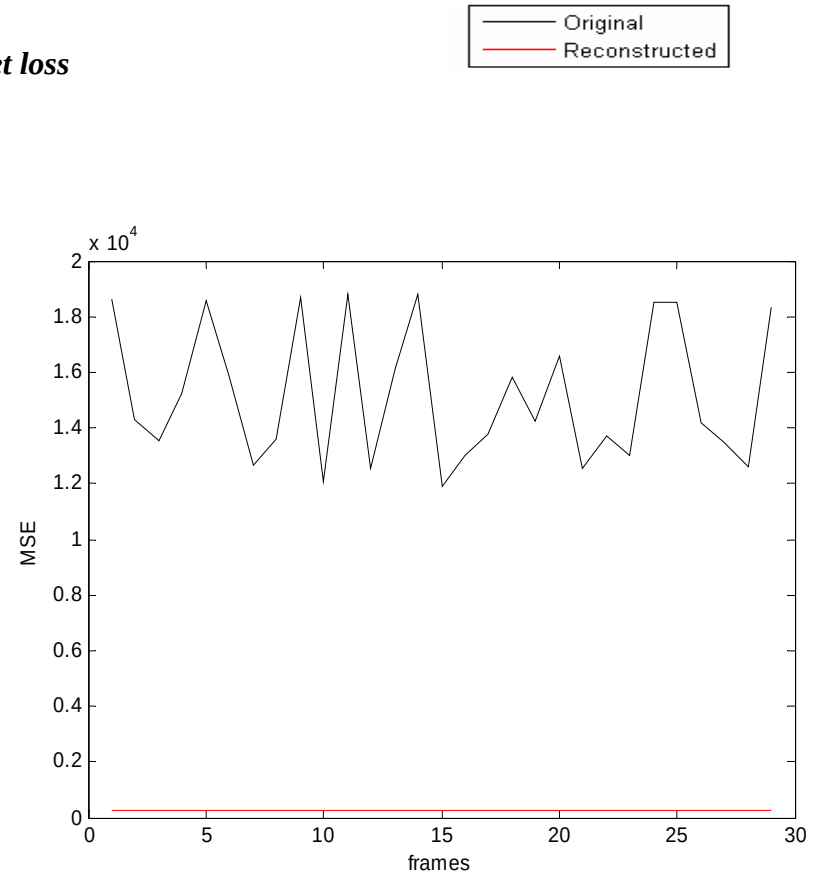
4.2. No Motion

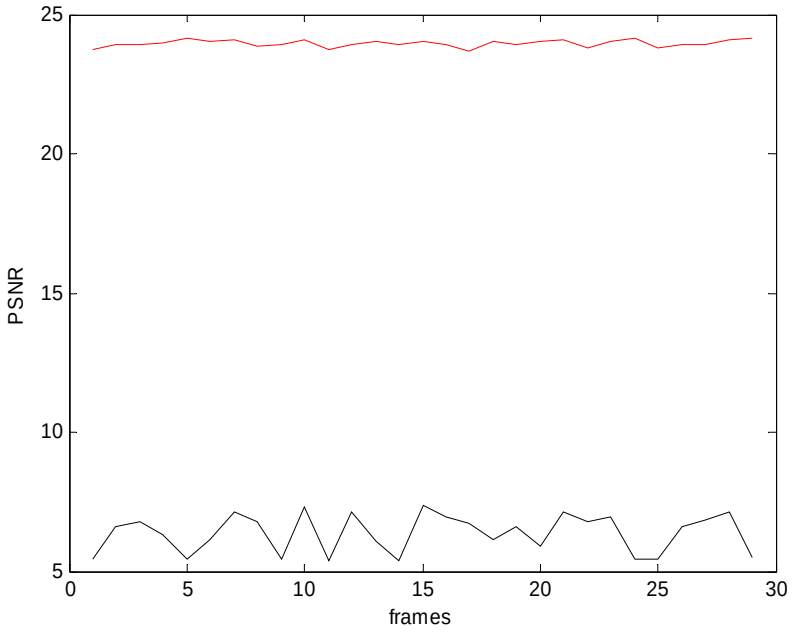
3% Packet loss



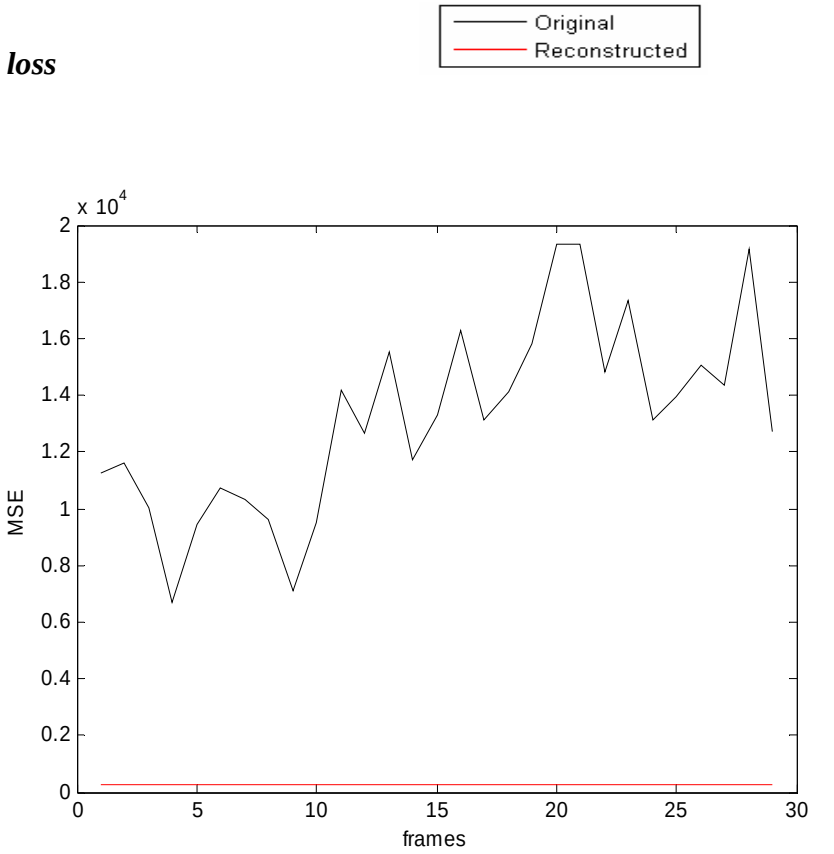


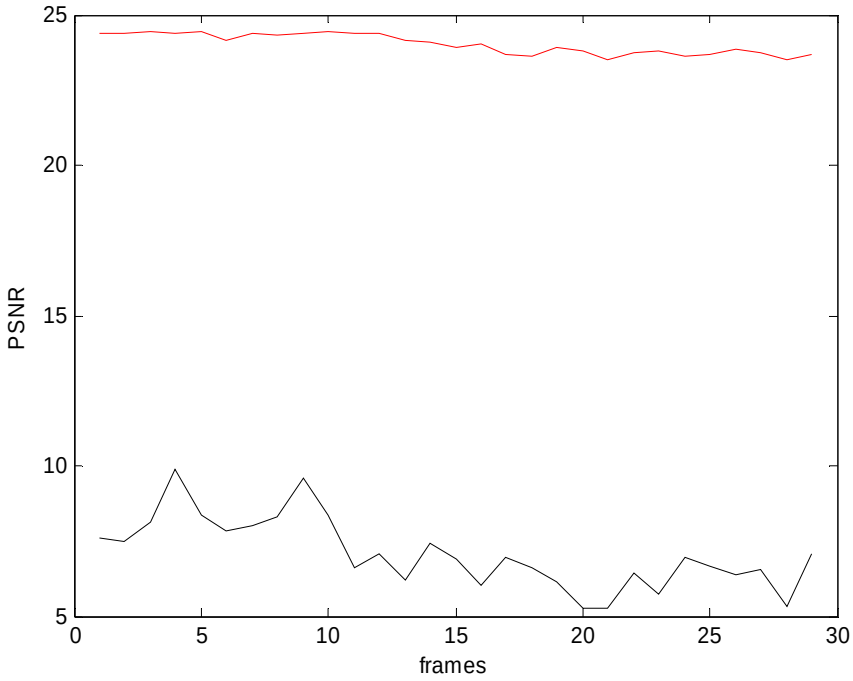
6% Packet loss





10% Packet loss





4.2.1. Comments on first group's results

The first group of results refers to the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR) of video with no motion. The error concealment algorithm performs very well in all three cases, namely 3%, 6% and 10 % packet loss. The PSNR is about 24 db and the RMSE is almost zero. On the other hand the values of the originally received video display a high deviation. Although it may appear strange, it is consistent with what is expected.

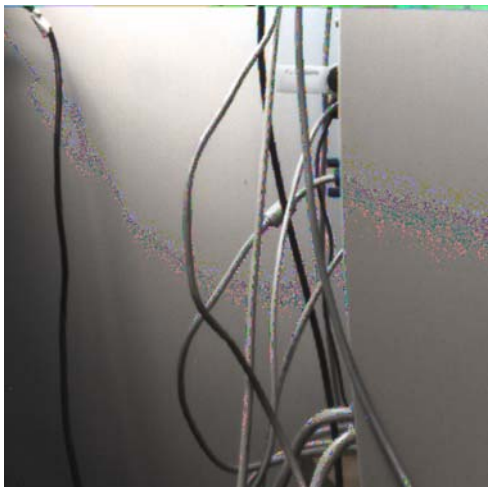
When a Restart Marker is lost from a frame, the frame is useless from that point on. Due to the decompression algorithm of the JPEG, when a Restart Markers is missing the decompression is stopped even if the rest of the image is valid. Therefore the degradation of the frame is irrelevant to the number of lost packets.

This explains the deviation of the values of the originally received video. The first missed Restart Marker is the only parameter controlling the quality of the received video.

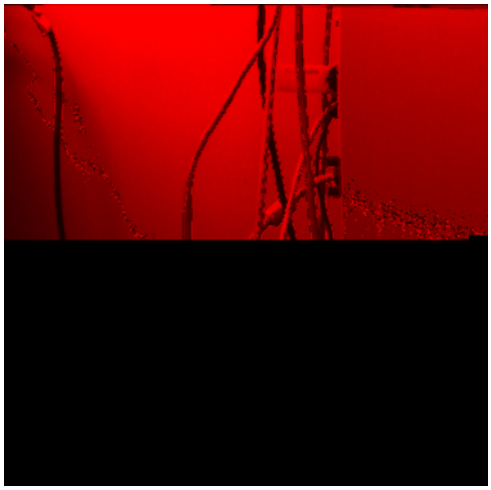
The frames in case 1, display how a packet loss can affect the quality of the received video. The originally received frame from the client contains structural information but little color information (only the red band). This is effect is cause by the way the JPEG image is compressed. The color components are encoded sequentially. When packet is lost, the rest of the frame is useless even if it is correct. Therefore only part of the color information is decompressed and displayed by the client.

The frames in case 2 display the results when the red and part of the green band components have been decompressed.

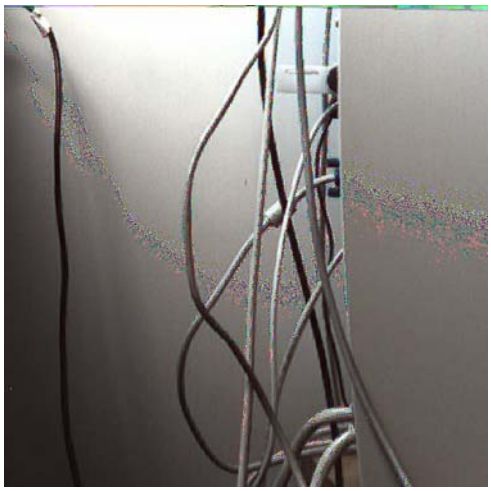
Frame from Server



Frame before Error Concealment



Frame after Error Concealment

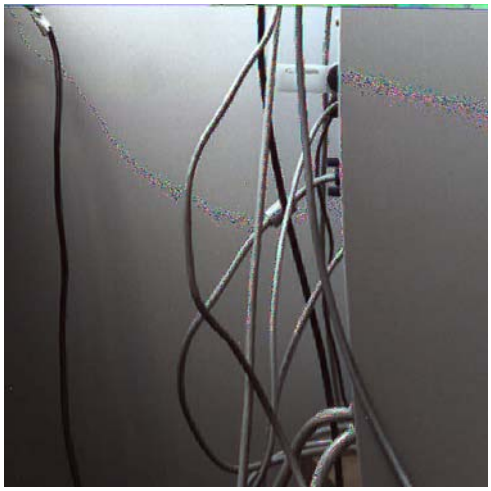


Case 1

Frame from server

Frame before reconstruction

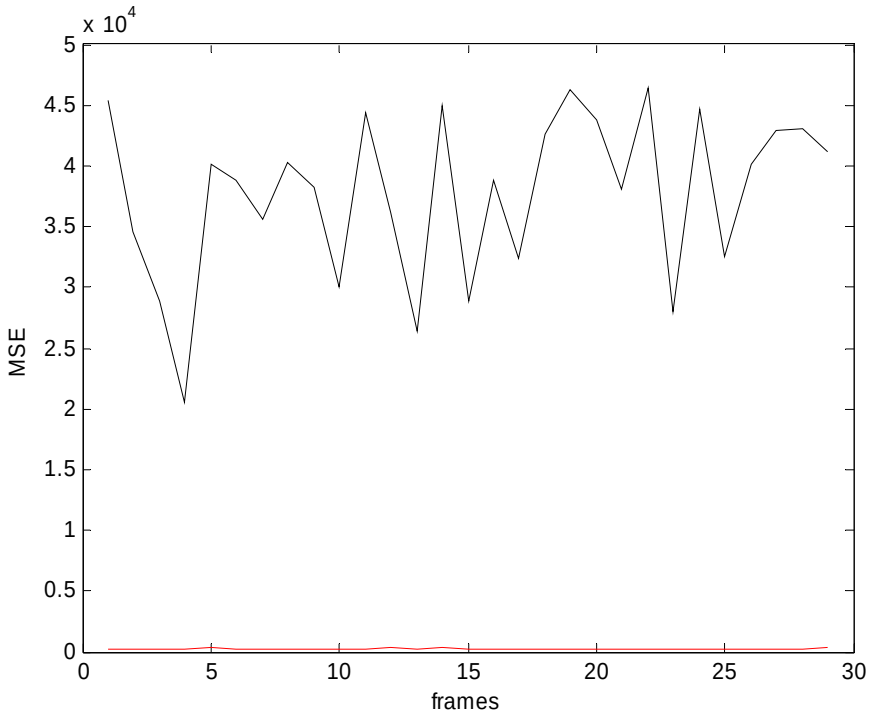
Frame after reconstruction

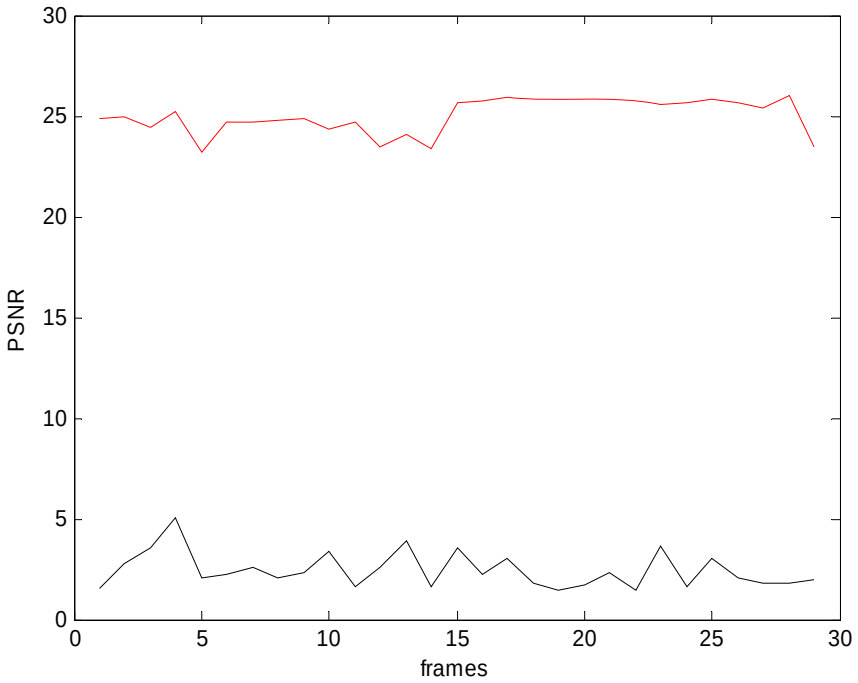


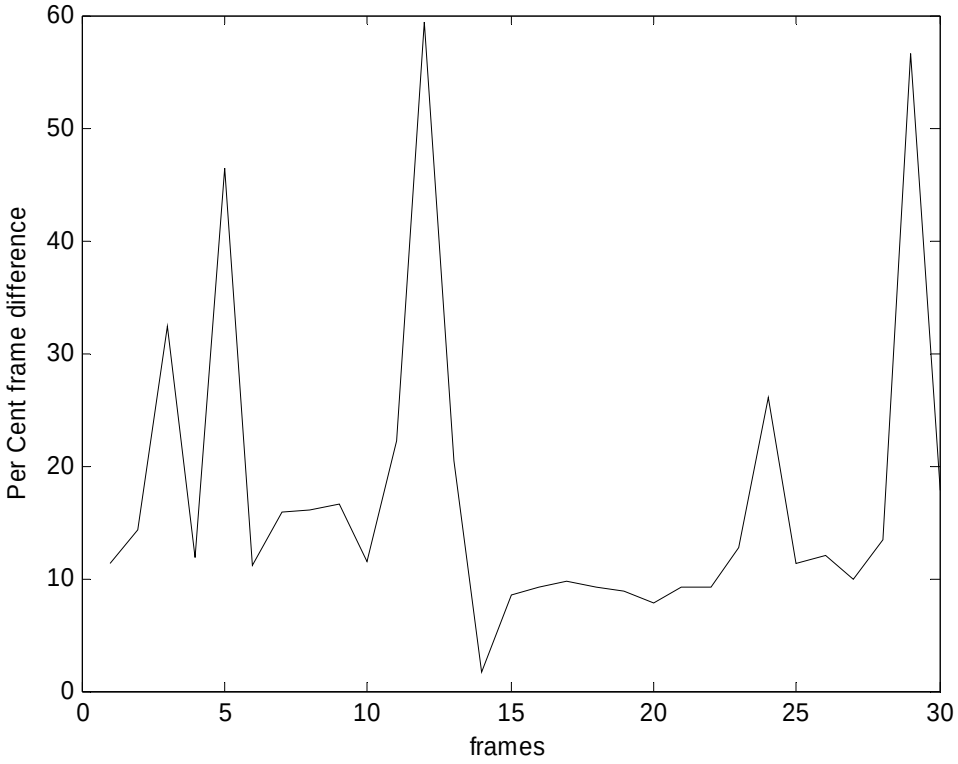
Case 2.

4.2.2. With motion

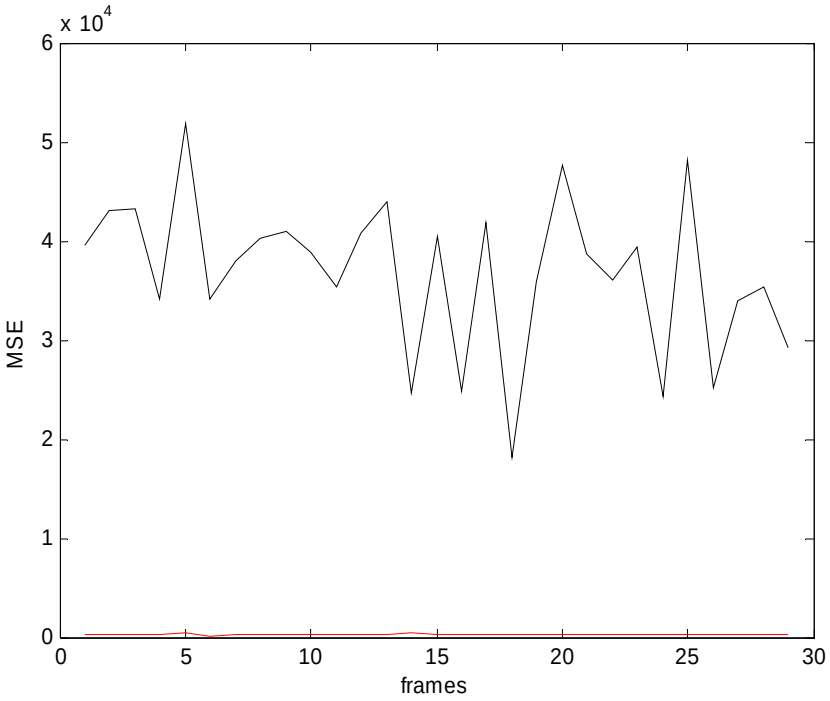
3% packet loss

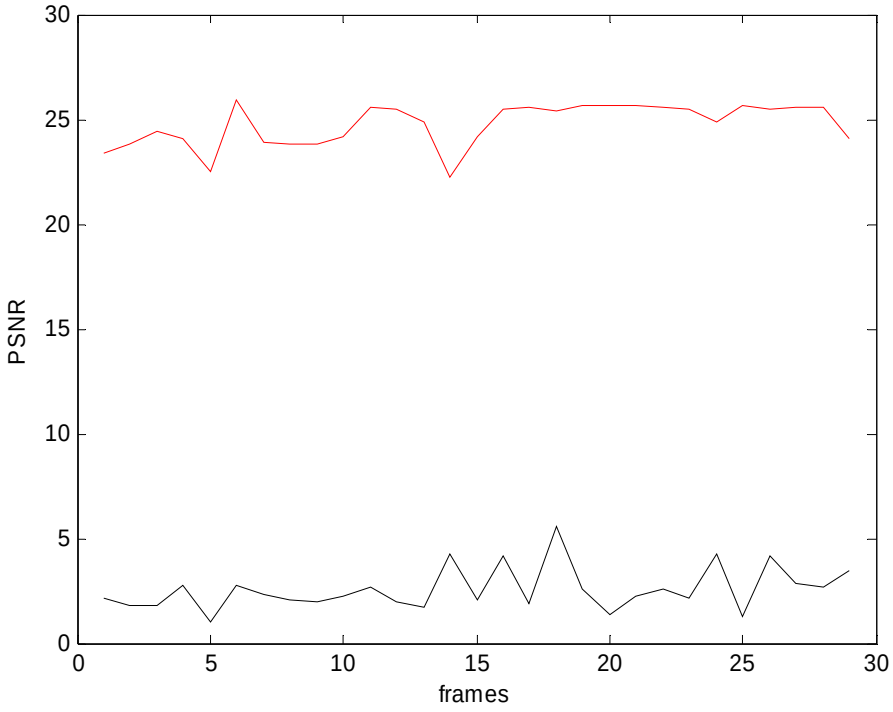


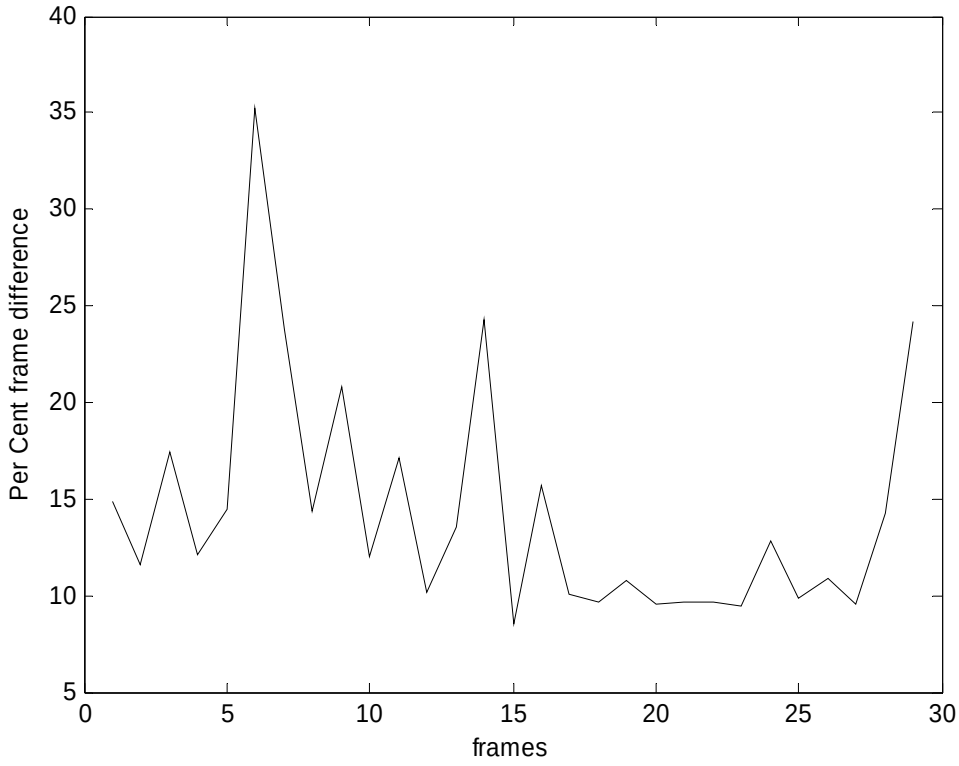



Per Cent frame difference
frames
0
10
20
30
40
50
60
0
5
10
15
20
25
30

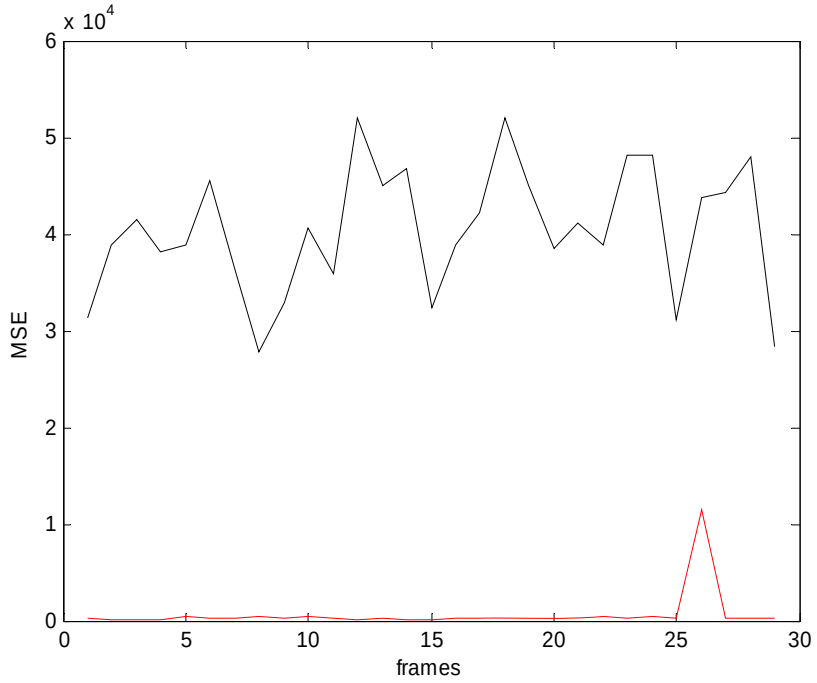
6% packet loss

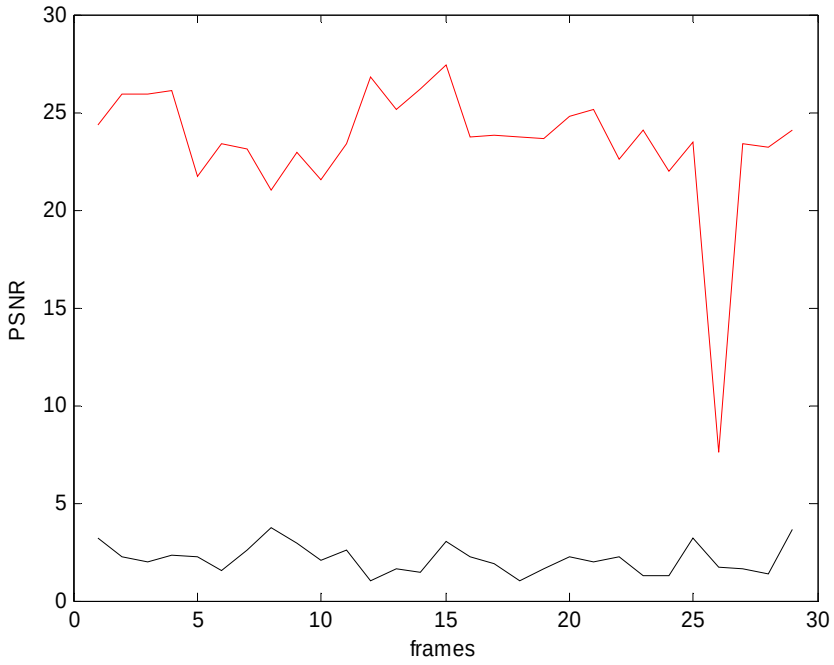


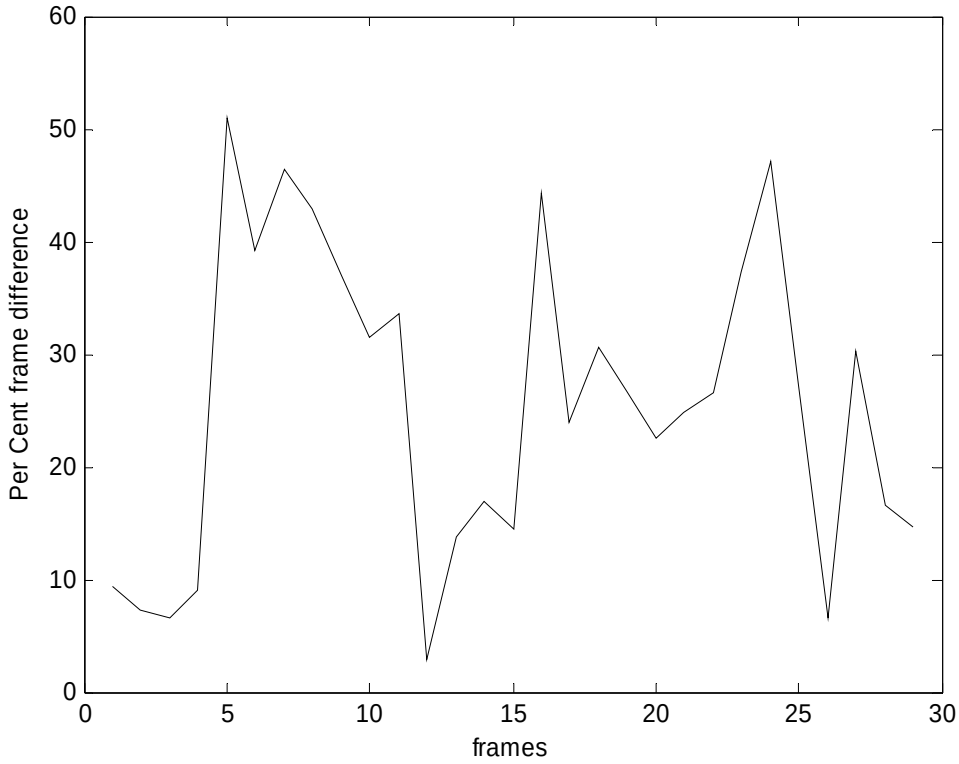



Per Cent frame difference
frames
40
35
30
25
20
15
10
5
0
5
10
15
20
25
30

10% packet loss






Per Cent frame difference
frames
0
10
20
30
40
50
60
0
5
10
15
20
25
30

4.2.3. Comments on second group's results

The second group of results examines the quality of the received video when there is some motion. The amount of motion is measured in terms of frame-to-frame difference (per cent). This metrics gives us a hint as to how much one frame has changed since the previous one. The error-concealment algorithm performs well, even in the case of motion. The motion of the video is not heavily reflected on the quality of the reconstructed image is terms of the PSNR. The PSNR values tend to "follow" the motion, but no in the same extend. The reason of this effect is that when the error concealment algorithm tries to estimate a missed packet, it uses as a reference the previously received corresponding frame. Since the current frame is not the same as the previous one, the quality of the reconstructed frame is lower.

The originally received video displays high deviation in its quality, deviation irrelevant to the number of packets lost, but consistent only to the first missed packet. For example, from the graphs when notice a high RMSE at frame 26. This value is so high because of the combination of high the motion in the previous frames and that one of the first packet of the image was lost. Missing one of the first packets implies that the received image became useless

CHAPTER 5

System Overview

Our system consists of a server and one or more clients. The server is a 5 frames per second colour camera connected with a Matrox ® DSP on a Pentium Xeon® computer with a one gigabit network card. The test clients are a Pentium Xeon computer a one gigabit network card and a Pentium 4 computer with a one gigabit network card. The client and the server are connected on a gigabit switch.
The grabbing and the display of the images are performed using the Matrox Mil® library.

Conclusion and Further Work

The error-concealment algorithm performs well when there is little or no motion in the video. In videos with more motion, the temporal redundancy quality of the video makes temporal based error-concealment perform badly. In order to achieve better quality of the received video, spatial based error concealment techniques must also be introduced.
Furthermore, a higher compression ration must be achieved due to the limitation of the available bandwidth. In any case, the computational complexity constrain must also be accounted for.
Surveillance oriented application must also take advantage of the growing area of the wireless networks.
Another standard close to the JPEG is the JPEG2000 [11]. JPEG2000 compression algorithm is based on wavelet transform, can achieve very good results, while it is keeping the computational complexity in low levels.
Another network protocol, the Real-time Transmission Protocol (RTP) [3] can also be used in transferring the MJPEG video bitstream. RTP is a general purpose video transport protocol, where the MJPEG video stream is registered as MJPEG load.

References

[1] The JPEG Still Picture Standard, Gregory K. Wallance, IEEE Transactions on Consumer Electronics.

[2] Detection and Correction of Transmission Errors in JPEG Images, Yh Han, JJ Leou - IEEE TRANS. CIRCUITS SYST. VIDEO TECHNOL. , 1998

[3] Computer Networking. A Top-Down Approach Featuring the Internet, James F. Kurose, Keith W. Ross, Pearson Education Inc

[4] Transporting Real-Time Video over the Internet: Challenges and Approaches. Dapen Wu, Yiwei Thomas and Ya-Qin Zhang, Proceedings of the IEEE, Vol. 88, No 12, December 2000

[5] JPEG File Interchange Format, Version 1.02, Eric Hamilton, September 1, 1992, C-Cube Microsystems

[6] A real-time algorithm for error recovery in remote video-based surveillance application, C Sacchi, F Granelli, CS Regazzoni, F Oberti - Signal Processing: Image Communication, 2002

[7] Multidescription Video Streaming with Optimized Reconstruction-Based DCT and Neural-Network Compensations, X Su, BW Wah - IEEE Transactions on Multimedia, 2001

[8] Information Technology – Digital Compression and Coding of continuous-time still image – Requirements and Guidelines, The International Telegraph and Telephone Consultative Committee, T.81, 09/92

[9] Joint Photographic Experts Group (JPEG) Image Compression for the National Imagery Transmission Format Standard, Department of Defence, Interface Standard MIL-STD-188-198A 15 December 1993,

[10] Communication in the Presence of Noise, Claude E. Shannon, the Proceedings of the IRE, vol. 37, no.1, pp. 10–21, Jan. 1949.

[11] http://www.jpeg.org/jpeg2000, site of the JPEG Committee