1996

Quality of Service Test by Codec Performance

Shunge Li

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Dawei Wang

Report Number:
96-071

QUALITY OF SERVICE TEST BY CODEC PERFORMANCE

Shunge Li
Bharat Bhargava
Dawei Wang

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR 96-071
November 1996

Quality of Service Test by Codec Performance*

Shunge Li Bharat Bhargava Dawei Wang
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{lis,bb,wangd}@cs.purdue.edu

Abstract

Quality of Service (QOS) test is a process of verifying the system's acceptability of a given QOS specification. Although the result of a QOS test depends on such system performance factors as the available resources, it is also influenced by the implementation of a codec scheme involved. In this paper, we argue that a good knowledge about the performance of the compressor and decompressor can provide solutions to QOS test. We present our approach to QOS test based on the performance of a JPEG compressor and decompressor and discuss various policies under which QOS negotiation process can follow if a QOS test fails.

1 Introduction

Quality of service (QOS) management and control for real-time multimedia applications over the high-speed networks has been a heavily researched topic ([16] [17] [5] [13] [4] [19] [2]). But an important question remains unanswered: given a QOS specification, how does the system know if it should accept or reject a QOS request? By QOS specification we mean a set of service requests in terms of QOS parameters, including loss rate, throughput, frame rate, response time, and presentation quality. The process of verifying the system's acceptability of a given QOS specification is a QOS *test*. A QOS test is *successful* if the system can meet the QOS specification requirements; otherwise, it is called *failed*. A QOS test is *conditionally failed* if it is failed but a negotiation of QOS requirements with the system is possible. A negotiated QOS specification can go through another round of QOS test to have its fate (success or failure) determined. The result of a QOS test can be affected by many factors, such as system resource availability and network transport protocols ([12] [7] [6]). We argue that besides these factors, the compression and decompression algorithms also have a big impact on QOS tests and that

*This research is supported by NSF under grant number NCR-9405931

their performance deserves to be evaluated closely for conducting QOS tests. Unfortunately, this impact has been largely ignored in the literatures. In this paper, we present an approach to QOS test based on the performance of a JPEG compressor and decompressor and discuss various policies under which QOS negotiation process can follow if a QOS test fails.

It has been widely accepted that compression techniques play a vital role in distributed multimedia applications. Not only the resulting data size, computational complexity, and information quality are determined by the codec (coding and decoding) schemes, but also the media specific protocols are heavily influenced by them ([14] [9]). For example, the error recovery strategy in a multimedia transport protocol would be different, depending upon whether or not the codec scheme involved has interframe dependency. It is well known that the overall performance of a distributed multimedia system, often abstracted as QOS parameters, is largely affected by the system resource availability and running environments. Besides these factors, it is also closely related to the type of the codec scheme involved, its implementation, and the settings of various codec scheme-related parameters. For example, the setting of a Q-factor parameter in JPEG compressor ([18]) influences not only the image size and compression time, but also the perceptual quality of the image.

Two well-known codec schemes: JPEG and MPEG ([8]) are widely used in distributed digital image and video applications. While JPEG was originally designed for still image compression, it can also be used for compressing motion pictures, in which case it is often cited as MJPEG (or Motion-JPEG). MPEG, on the other hand, is the default codec scheme for motion pictures in many applications. MJPEG/JPEG and MPEG have very similar codec algorithms except that MPEG computes motion compensation components among consecutive frames whereas MJPEG encodes consecutive frames individually. The fact that MPEG takes advantage of the temporal similarity among consecutive frames gives it a much higher compression ratio than MJPEG, but it increases the interframe coupling, which may incur extra overheads in error recovery during unreliable transmission. Besides, more buffer spaces have to be maintained on both ends of transmission, complicating the buffer management task.

Contrarily, because MJPEG does not have interframe dependency, the error recovery scheme can be very simply due to the fact that any lost/corrupted frame does not have impacts on other frames. Moreover, since there is no need for motion compensation, which is a computation-intensive task, MJPEG can be fast. These characteristics of MJPEG make it quite attractive in some cases where simple and fast error recovery scheme is preferred. Of course, the major disadvantage of MJPEG when compared to MPEG is that the compressed data set from MJPEG is considerably larger than that from MPEG. This implies a bigger network bandwidth consumption and storage space requirement. Poor video quality will be exhibited if higher compression ratio for MJPEG is achieved. It is clear that for different application QOS requirements, MJPEG and MPEG can each provide better performance than the other. Therefore, rather than naively choosing one over the other, the conditions under which MJPEG can outperform MPEG, or vice versa, must be investigated. In this paper, we

focus on the performance evaluation of JPEG compression algorithm due to its applicability not only in the area of digital video but also in the area of image database and digital library.

Since JPEG became a standard for image compression, people have been working on the improvement of its performance for various applications by developing more efficient JPEG algorithms and by implementing JPEG software packages. These software packages offer various parameters that can be set by applications. It is up to the application developers to figure out the "correct" settings for a certain application. Since no guidelines on how the setting of each parameter would precisely affect the performance, their efforts are based solely on their intuitions and/or general knowledge about JPEG standard and thus are not fundamentally sound. Moreover, despite the close ties between the settings of these JPEG parameters and QOS of the applications that use JPEG, few address, explore, or even realize its importance. No specific details about how to perform application QOS control based on the observations from experimentation of different JPEG parameter settings have been presented in the literature. This topic has been largely ignored and many important research questions regarding the performance issues remain unanswered. Following is a list of typical questions of this topic:

- When to use what (parameters) to achieve best possible performance or to conform the QOS requirements? For instance, when it is good to generate and use images' own Huffman Tables and when it is good enough to use the default ISO-defined Huffman Tables?
- What are the trade-offs among such parameters as presentation quality of images, compression time, compressed data size, storage cost, transmission time, end-to-end delay, and decompression speed? What are the criteria for the trade-offs? How to achieve a good balance among them?
- Given a QOS specification such as the upper-bound of response time and/or lower bound of presentation quality, how to set the JPEG parameters to ensure that this specification will be satisfied? For example, which value of Q-factor is good enough for a specific response time bound and presentation quality bound? If the QOS test fails, how to adjust the JPEG parameters to accommodate the negotiated QOS?
- What are the relationships between application QOS and system resource constraints? Once the knowledge about these relationships are known, how to adapt the application to the environmental changes automatically?

Partly motivated by these questions, this paper intends to provide some solutions on how to select JPEG parameters through experimentation. In addition to this intention, this paper will formulate the QOS test problem and address its importance in QOS control for distributed multimedia systems.

This paper is organized as follows. Section 2 revisits the basics of JPEG baseline sequential compression scheme. Section 3 presents our experiments on the performance of a commercial JPEG implementation and our observations. The formulation of the QOS test problem and its

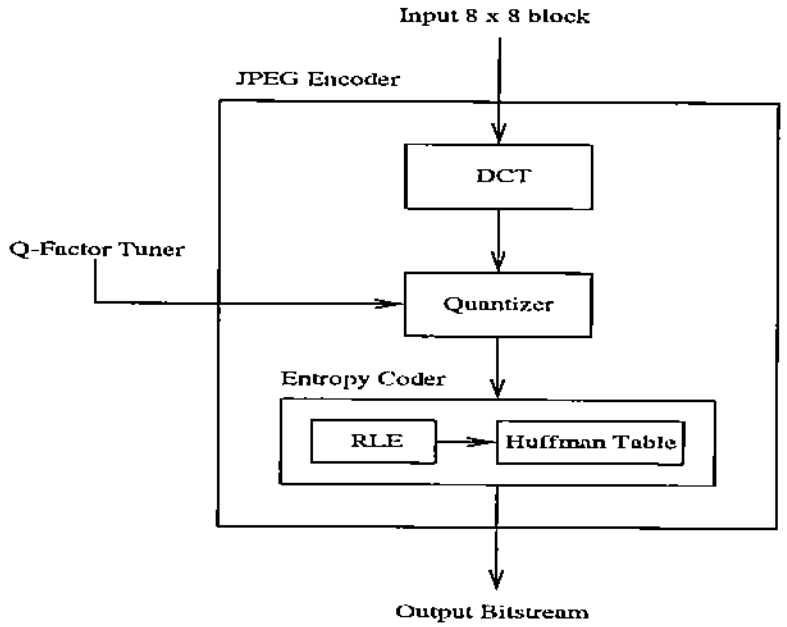

Figure 1: JPEG Baseline Sequential Compressor

algorithm will be given in section 4. Additionally, how to perform application QOS negotiation based on the performance evaluation of a JPEG compressor will be discussed. Principles that may be followed during QOS negotiation will be proposed. An adaptive scheme for computing Huffman Tables in MJPEG will be proposed. Concluding remarks are given in section 5.

2 JPEG Compression Scheme Revisit

2.1 JPEG Baseline Sequential Compressor

JPEG compression standard ([18]) was jointly proposed by ISO and CCITT, and was originally designed for still image compression. There are two standards involving JPEG compression schemes: JPEG lossless scheme and JPEG baseline sequential scheme. As the names suggest, the former is lossless compression scheme and the latter is a lossy one. While lossless compression scheme can provide high quality images, it results in low compression ratio. On the other hand, though JPEG baseline sequential compression scheme may cause some information loss, it can achieve a much higher compression ratio, resulting in tremendous storage savings as well as network transmission time decrease, without too much quality degradation. Since many multimedia applications such as video conferencing and video-on-demand can tolerate a certain amount of information loss, lossy compression schemes have wider use than their lossless counterparts.

A JPEG baseline sequential compressor consists of three steps (figure 1):

1. DCT (Discrete Cosine Transform): This step performs a mathematical operation that converts a block of image samples from the spatial domain to the frequency domain. In JPEG, the input to the DCT module is an 8 x 8 matrix whose values represent brightness levels at particular x, y coordinates, and the output is an 8 x 8 matrix whose values represent relative amounts of the 64 spatial frequencies that make up the input data's spectrum. One property of DCT that has been explored in the JPEG scheme is its frequency distribution: low frequency components occupy the upper-left corner of the matrix and high frequency components the lower-right corner of the matrix. Typically low spatial frequencies outweigh high spatial frequencies, namely, low frequency components have relatively larger coefficiencies. As a result, most of the values in the output matrix, outside of those in the upper-left corner, will have values close to 0 and will end up not being encoded.

2. Quantization: The purpose of quantization is to scale the resulting values from DCT into ones within a range of [0, 255]. To achieve this, a *quantization table*, which is an 8 x 8 matrix, is used. This table is so designed to recognize the frequency distribution resulted from DCT, that is, it provides non-linear scaling factors for different portion of the matrix. More scaling levels are provided for upper-left corner than for lower-right corner. The precision of the quantization is adjustable through a parameter known as quantization factor or Q-factor for short. It is the Q-factor that can control the amount of the information loss (therefore the quality of an image) and the compression speed.

3. Entropy Coding: This step creates actual JPEG bitstream. The entropy coder encodes the quantized block of data in a diagonally zigzag fashion originating from the upper-left corner to ensure that the encoder will encounter all nonzero values in the block as early as possible. As the encoder works all the way through the values in the zigzag order, it records three pieces of information each time it encounters a nonzero value: the occurrence of zeros it passed over before finding the nonzero value, the number of bits it will take to encode the nonzero value, and the value itself. The first two pieces of information are considered as a pair. To generate a bitstream for these three pieces of information, the encoder consults a Huffman Table to find the bit sequence that represents the pair of the first two pieces and encodes the third piece using a variable-length code. Because the entropy coder generates variable length codes, it is often called *Run-Length Encoder*, or RLE. The RLE continues this process until all the remaining values in the block are zeros, at which point, it marks the bitstream with a special end-of-block bit sequence.

For more details about JPEG algorithm, please consult [18].

2.2 XIL Implementation of JPEG Baseline Sequential Compressor

Our work has been based on the SunSoft's JPEG implementation, available in the XIL library package ([15]). The XIL Imaging Library is a product of SunSoft, Inc. It provides a set of key functions from the fields of image processing and digital video. It serves as an application programming interface (API) that has special characteristics. On one hand, it hides the hardware-specifics from applications by directly dealing with hardware dependencies. On the other hand, it provides support for a broad area of applications such as document imaging, facsimile applications, and digital video. Therefore, it is a layer between the applications and the hardware devices.

The XIL library is available only on Solaris 2.3 operating system or up. It is written in C programming language, so is its API. Therefore, applications wishing to use XIL library are recommanded to be implemented in C.

XIL library is a general purpose library in that it does not make any assumptions for applications and can work on a variety of environments. To achieve this generality, it provides a lot of programmable parameters which can be set by the application programs, depending upon the environments on which the XIL library and the applications intend to run. For example, in JPEG baseline codec implementation, it allows an application to choose, among several options, an appropriate quantization table, appropriate values of compression quality (also known as Q-factor) and decompression quality, and whether the default Huffman Tables supplied by the XIL library or the optimal Huffman Table generated on-the-fly are preferred, etc. These options serve as performance *tuners* which, when set differently, can lead to different system performance.

With such a variety of options supplied by the XIL library, which may themselves interrelate to each other, it is less intuitive for application developers to figure out the "right" settings in order to achieve best performance possible or to satisfy QOS requirements. Our experiments intended to find the conditions under which a set of settings of options are "right" for a certain applications in terms of QOS specifications or best performance possible.

3 Experiments

3.1 Experiment Setup

We conducted the experiments between a SunSparc-5 workstation and a SunSparc-10 workstation interconnected via a 10Mbps Ethernet link. Both workstations were equipped with SunVideo cards and color video cameras. Solaris 2.4 operating system was running on both machines. The JPEG baseline sequential compressor was implemented using XIL library ([15]). All the data were measured on the SunSparc-5 workstation.

3.2 Notations

To simplify discussion, we denote many JPEG parameters with symbols, as shown in table 1.

Table 1: Notations for JPEG Parameters

Notation	Name	Meaning
CR	Compression Ratio	$\frac{S_0}{S_1}$, where S_0 is the size of an uncompressed frame and S_1 the size of a compressed frame
T_c	Compression Time	Time spent in compression
T_d	Decompression Time	Time spent in decompression
T_t	Transmission Time	Time spent in data transmission
CQ	Compression Quality	A number $\in$ [1, 100] indicating the balance between quality and compression speed
DQ	Decompression Quality	A number $\in$ [1, 100] indicating the balance between quality and decompression speed
PQ	Presentation Quality	A number representing subjective image quality level
HT	Huffman Table	A table used in entropy coding
N_f	Number of Frames	The number of consecutive frames
T_s	Service Time	Time duration from when the server receives a request to when the user receives a reply
T_r	Response Time	Time duration from when a user sends a request to when the user receives a reply

The dependency relationships among these parameters are very complex, as indicated by a dependency graph (figure 2). The nodes stand for the parameters, whereas the arrowed arcs stand for the dependency relations with the directions of arrows towards depending parameters.

Input parameters are those with their arrowed arcs pointing to other nodes and without being pointed to by others. CQ, DQ, N_f, HT, and S_0 are input parameters in the graph. Output parameters are those without arrowed arcs pointing to other nodes but being pointed to by others. PQ, T_s, and CR are output parameters. The remaining parameters are both output parameters in some cases and input parameters in other cases. They include S_1, T_c, T_d. With a dependency graph, it is easy to identify input parameters, output parameters, and others.

In reality, PQ and T_s are input parameters in a QOS specification. They serve as upper or lower bounds in the relations. Our task is to find a set of PQ and T_s values which are bounded by the specified input values of PQ and T_s. These dependency relationships will become clear in the next subsections.

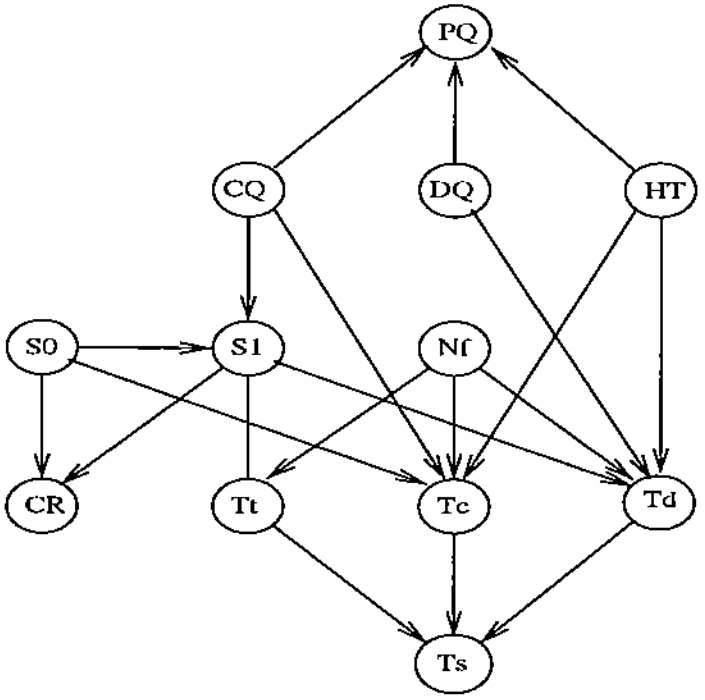

Figure 2: Dependency Graph of JPEG Parameters

3.3 Performance Evaluation

3.3.1 Experimenting with Huffman Tables

A Huffman Table is used in the entropy coding phase in the JPEG or MPEG algorithms. There are traditionally two approaches to computing and using the Huffman Tables. One is static, that is, a fixed Huffman Table is used for all the images involved. This table can be one of the defaulted ISO-defined Huffman Tables, or can alternatively be computed based on any single image. The other is dynamic, that is, a Huffman Table is generated on-the-fly for each image.

Figure 3 compares the compression time when using a default Huffman Table with that when computing an optimal Huffman Table. Their difference, which is almost a constant in [0.20, 0.30] seconds with respect to CQ, indicates the overhead involved in an optimal Huffman Table computation. Note that although this number (t_{HT}) is a constant, it is a machine-dependent value.

The parameter N_f extends the problem to a new dimension: time. Instead of considering one image, we now consider a sequence of images and are interested in knowing the overhead of the storage space a Huffman Table consumes. Figure 4 shows how the compression ratio changes as a function of N_f for a fixed value of compression quality.

Clearly, for a fixed value of compression quality, compression ratio increases as N_f in-

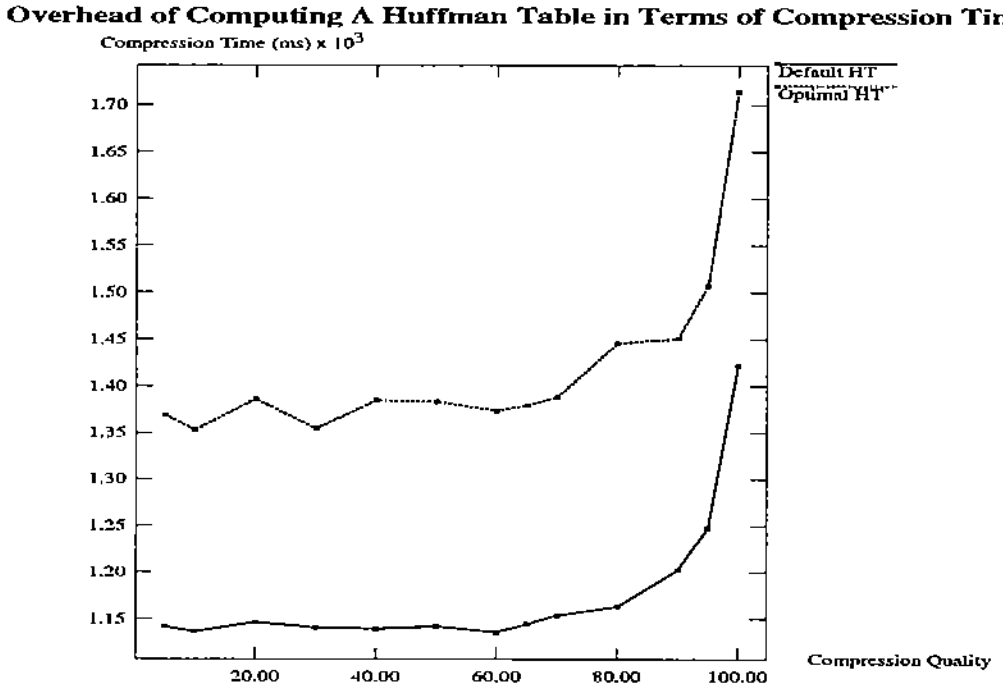

Figure 3: Overhead of Computing An Optimal Huffman Table

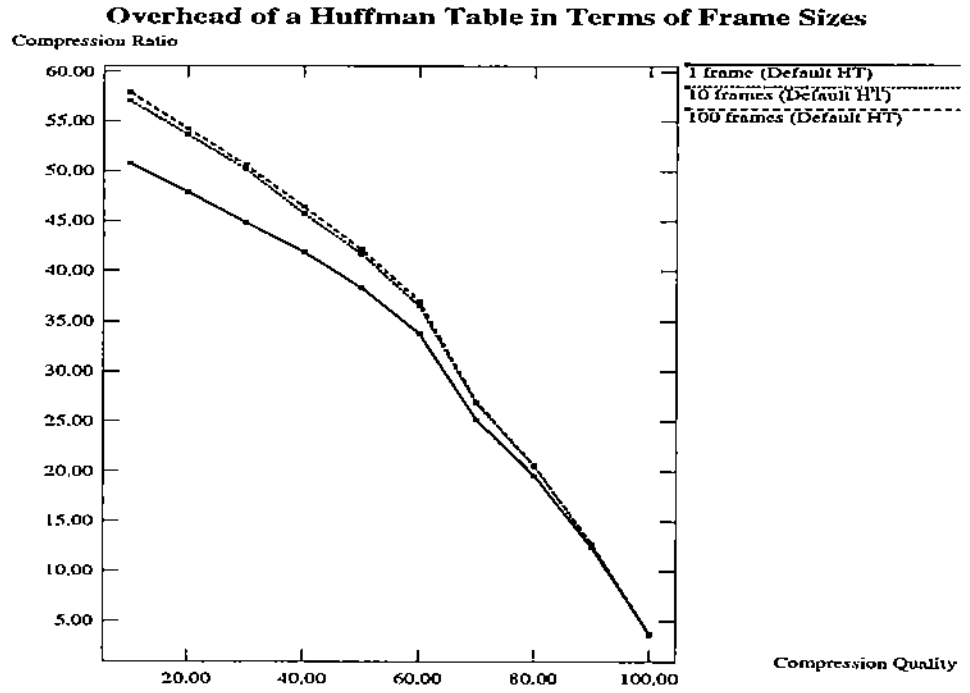

Figure 4: Number of Frames vs Huffman Tables

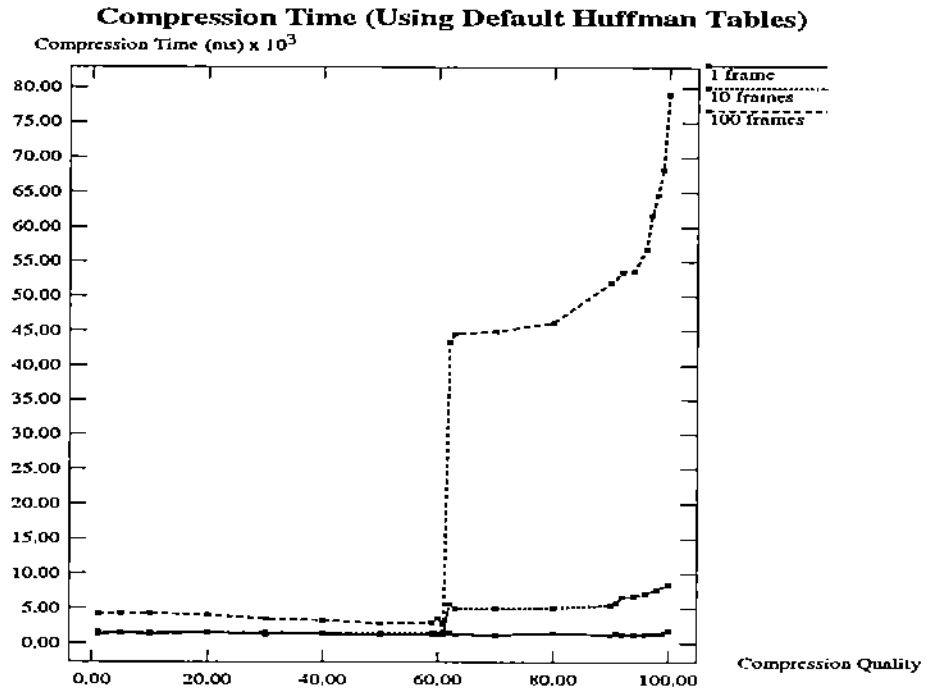

Figure 5: Compression Time as a Function of Compression Quality

creases. Furthermore, the closer to 1 the values of N_f, the bigger the difference between their compression ratios. This can be attributed to the overhead of the Huffman Table. Since only one default Huffman Table is used regardless of the number of the frames, the overhead of the Huffman Table in terms of the storage space it consumes will be amortized among all the frames, and therefore the more the frames, the less the amortized overhead per frame.

3.3.2 Compression Time as a Function of Compression Quality

CQ is a parameter that can be set by applications to adjust the compression time and the resulting image quality. CQ tells the compressor how it should handle the trade-off between image quality and compression. It can be set to any value between 1 and 100. Setting CQ to 100 is a request that the compressor produce very high quality images, even though this will mean a lower compression ratio. A setting of 1 tells the compressor to increase its compression ratio, even though the result will be lower image quality. By default, CQ is set to 50.

The setting of CQ controls image quality and the compression ratio by determining a scaling factor the compressor will use in creating scaled versions of its quantization tables. These scaled tables are used during compression. A CQ value of 50 results in a scaling factor of 1; that is, the scaled tables are identical to the original tables. A CQ value greater than 50 results in a scaling factor that is less than 1, and a value less than 50 results in a scaling factor greater than 1.

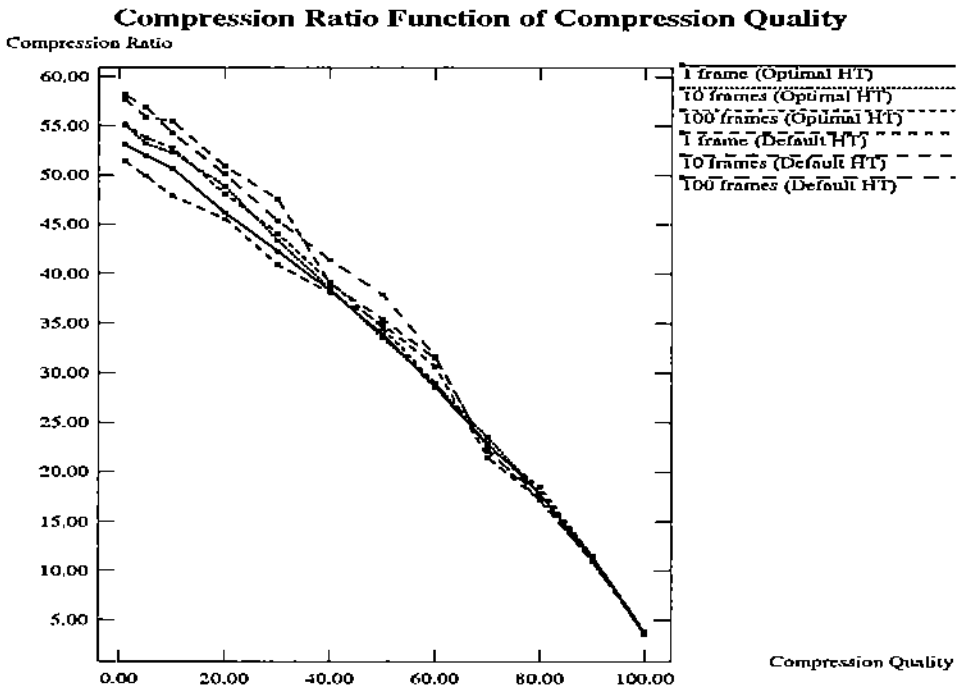

Figure 6: Compression Ratio Function

Figure 5 shows the effects of different CQ values and N_f on compression time. It can be learned from this figure that when $N_f = 1$, T_c is almost a constant; when $N_f \gg 1$, the compression time is a non-linear function of CQ. It takes approximately the same time to compress an image when CQ is set in the range of 1 to 60. A threshold value of CQ exists around 62, which leads to a significant increase of compression time. Similarly, the compression time has a big leap when CQ is about 90.

This observation is quite useful because we can save a lot of compression time by simply setting CQ to be 60 instead of 70 without greatly degrading the perceptual image quality (the direct relationship between the value of CQ and the perceptual image quality is shown below). Setting CQ less than 60 gains nothing in terms of compression time, compared to that when CQ is set to 60.

For the purpose of later discussion, we denote this function as

$$T_c = T_c(CQ, S_0, N_f, HT) \tag{1}$$

3.3.3 Compression Ratio as a Function of Compression Quality

Figure 6 quantitatively tells us how different CQ values affect the compression ratio, which is directly related to the compressed image size.

It is observed that all the curves in figures 6 converge to one line as CQ increases to be greater than 70. Furthermore, CR is approximately a linear function of CQ, which has the following form,

$$CR \simeq \mathcal{F}(CQ) = K \cdot (CQ - CQ_{min}) + CR_{max} \tag{2}$$

where K is a constant and can be calculated as follows:

$$K = \frac{CR_{max} - CR_{min}}{CQ_{min} - CQ_{max}} \tag{3}$$

where $CQ_{min} = 1$, $CQ_{max} = 100$, $CR_{max} \in [51, 59]$, and $CR_{min} \simeq 3.7$, based on the experimental data shown in figure 6. Therefore, K is a negative number in the range of [-0.56, -0.48].

The value of CR is solely determined by CQ and is almost not affected by the way Huffman Table is computed, that is, whether the default Huffman Table or the optimal Huffman Table is used, the values of CRs are the same, as long as they correspond to the same CQ. Another observation is that CR is independent of N_f. Since $CR = \frac{S_0}{S_1}$, we have

$$S_1 = \mathcal{G}(CQ) = \frac{S_0}{K \cdot (CQ - CQ_{min}) + CR_{max}} \tag{4}$$

as indicated by figure 7.

Figure 8 shows the average size of a compressed frame as a function of CQ and N_f. The fact that all the six curves almost coincident implies that S1 is irrelevant of the value of HT and the size of compressed frames is proportional to N_f. This means each compressed frame has almost the same size. This fact can be expressed as follows:

$$\mathcal{G}(CQ, N_f) = N_f \cdot \mathcal{G}(CQ, 1) = \frac{N_f \cdot S_0}{K \cdot (CQ - CQ_{min}) + CR_{max}} \tag{5}$$

We must point out that the picture of equation (4) is very close to figure 8. This evidence strongly justifies the correctness of equation (2).

3.3.4 Compression Time as a Function of Compression Ratio

The size of a compressed image is one factor that determines the data transmission time as well as the storage time. For a specific application, uncompressed images usually have the same size (unless the resolution or physical size is changed). Therefore, compression ratio is reversely proportional to the size of compressed images.

The relationship between the compression time and the compression ratio, which reflects the space-time trade-off in the compression process, can be derived from figure 5 and figure 6. From equation (2), we get

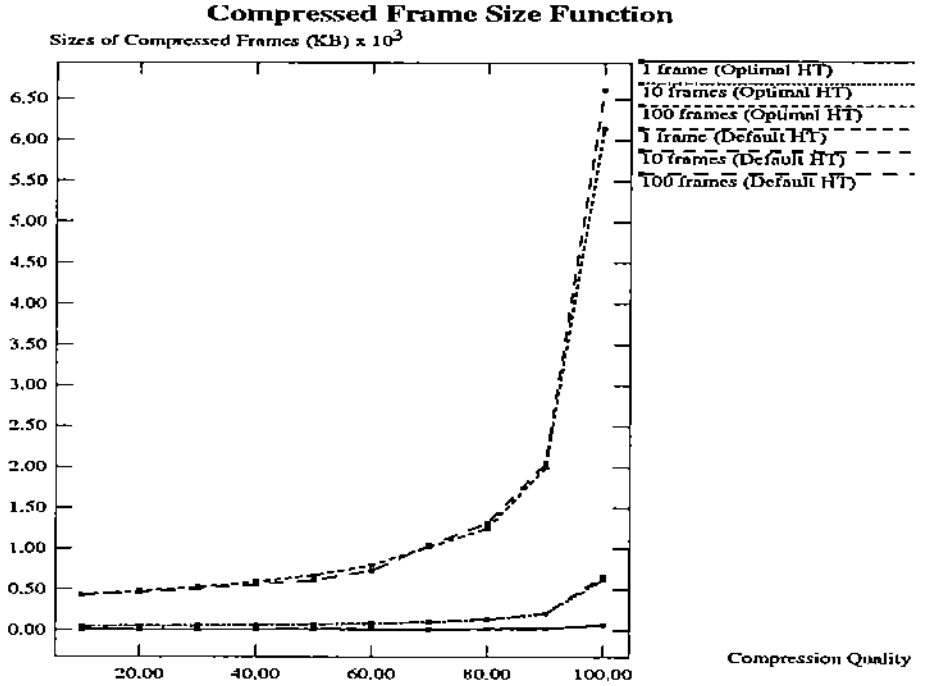

Figure 7: Compressed Frame Size Function

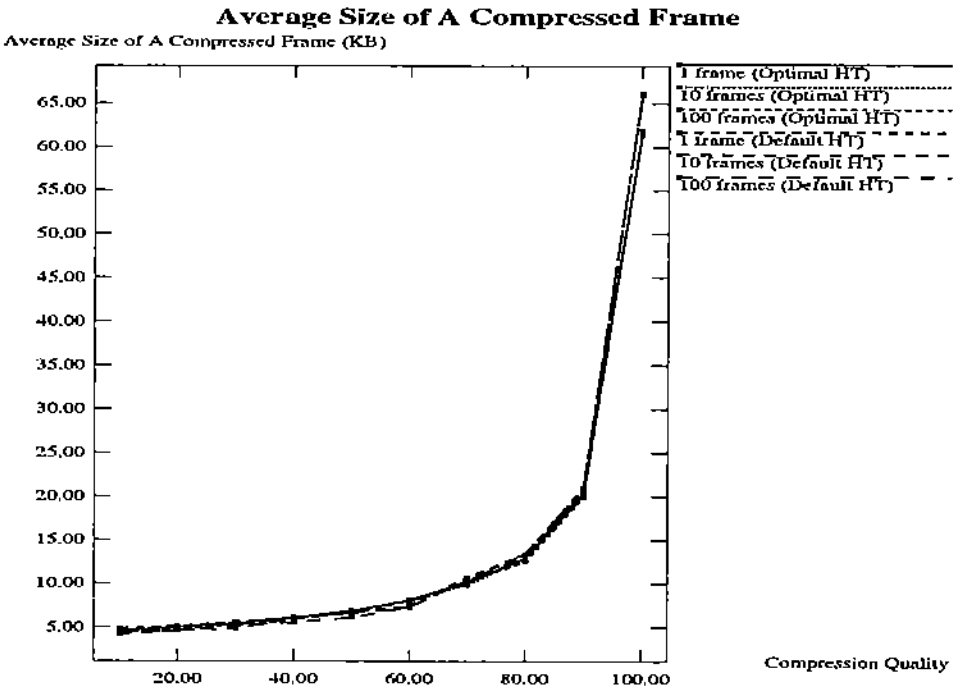

Figure 8: Average Single Compressed Frame Size Function

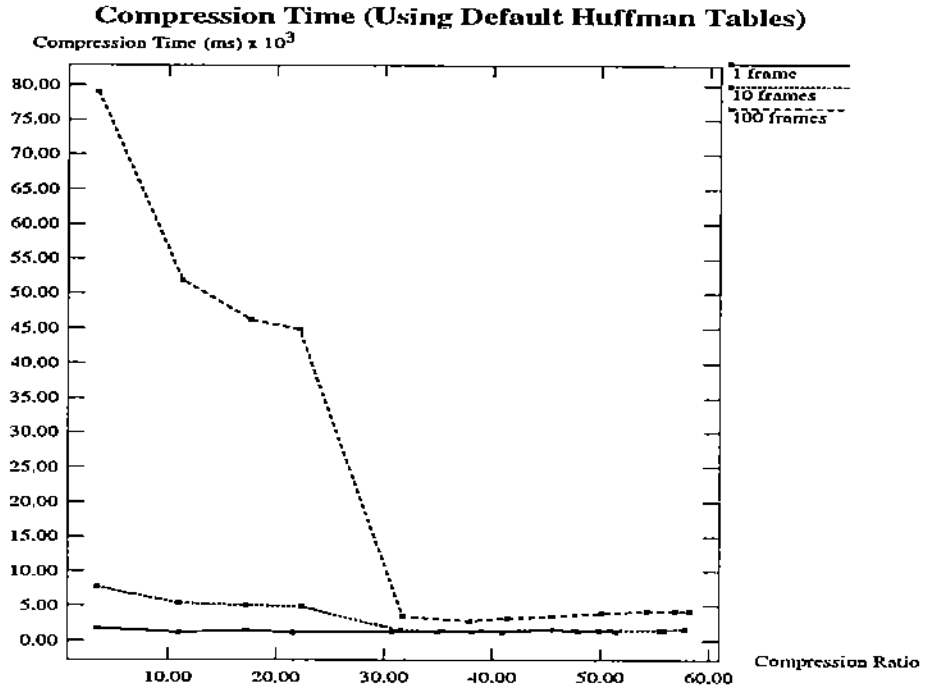

Figure 9: Compression Time as a Function of Compression Ratio

$$
\begin{aligned}
CQ &= \mathcal{F}^{-1}(CR) = K^{-1} \cdot (CR - CR_{max}) + CR_{min} \\
&= K^{-1} \cdot CR + (CQ_{min} - \frac{CR_{max}}{K}) \\
&= K^{-1} \cdot CR + (CQ_{min} - \frac{CR_{max}}{CR_{max} - CR_{min}} \cdot (CQ_{min} - CQ_{max}) \\
&= \frac{CQ_{min} - CQ_{max}}{CR_{max} - CR_{min}} \cdot CR + \frac{CQ_{max} \cdot CR_{max} - CQ_{min} \cdot CR_{min}}{CR_{max} - CR_{min}} \qquad (6)
\end{aligned}
$$

Combining equations (6) and (1), we get another expression of compression time, namely, $T_c = T_c(CR, S_0, N_f, HT)$, as shown in figure 9.

3.3.5 Decompression Time as a Function of Decompression Quality

Similar to CQ, DQ provides a hint to the decompressor concerning how it should handle the trade-off between the quality of decompressed images and speed of decompression. The decompressor is free to ignore this hint. DQ is an integer in the range of 1 to 100. A setting of 100 is a request for a high level of image quality, and a setting of 1 is a request for a high playback speed. By default, DQ is set to 100.

Theoretically, the JPEG decompressor increases playback speed by decreasing the number of quantized coefficients it uses in reconstructing an image. It drops high-frequency coefficients

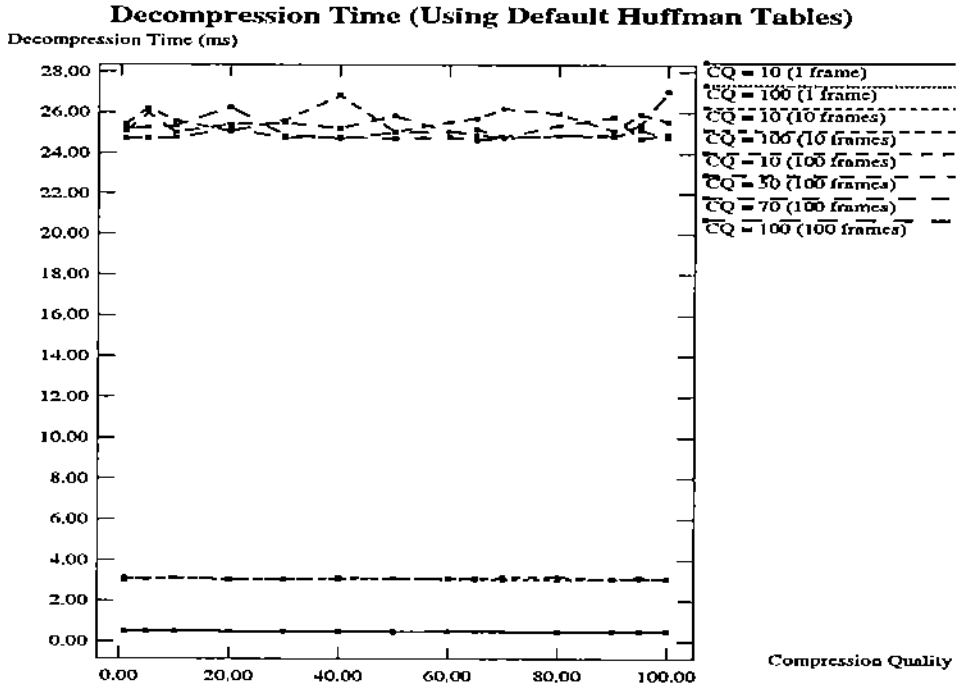

Figure 10: Decompression Time as a Function of Decompression Quality

first. In practice, the decompression speed, though varying within a certain range, is not largely affectly by the value of DQ. Figure 10 shows this observation.

3.3.6 Data Transmission Time as a Function of Data Size

The time spent in data transmission can be affected by many factors, among which the most important ones are the data size, network capacity, distance between two communicating ends, and queuing delay at the intermediate routers. For the purpose of discussions in this paper, we focus on the effects of data size on transmission time, and ignore other factors for a moment. Therefore, the transmission time T_t can be denoted as a function of data size, that is, $T_t = T_t(S)$. It is a monotonically increasing function, as shown in figure 11.

The figure reveals that for the same size of data, the transmission time over Wide Area Network (WAN) is $1 \sim 2$ magnitude higher than that over Local Area Network (LAN). This difference suggests that it is better to compress the data before transmitting it over WAN because the savings in transmitting data over WAN can compensate the time spent in compression and decompression, and that sometimes it may be better *not* to compress the data but just transmitting the data over LAN because compared to the short latency in LAN, compression and decompression times may not be sufficiently compensated.

Similarly, for fixed destination, depending upon the data sizes, there are cases where compression, transmission, and decompression is faster than simple transmission without compres-

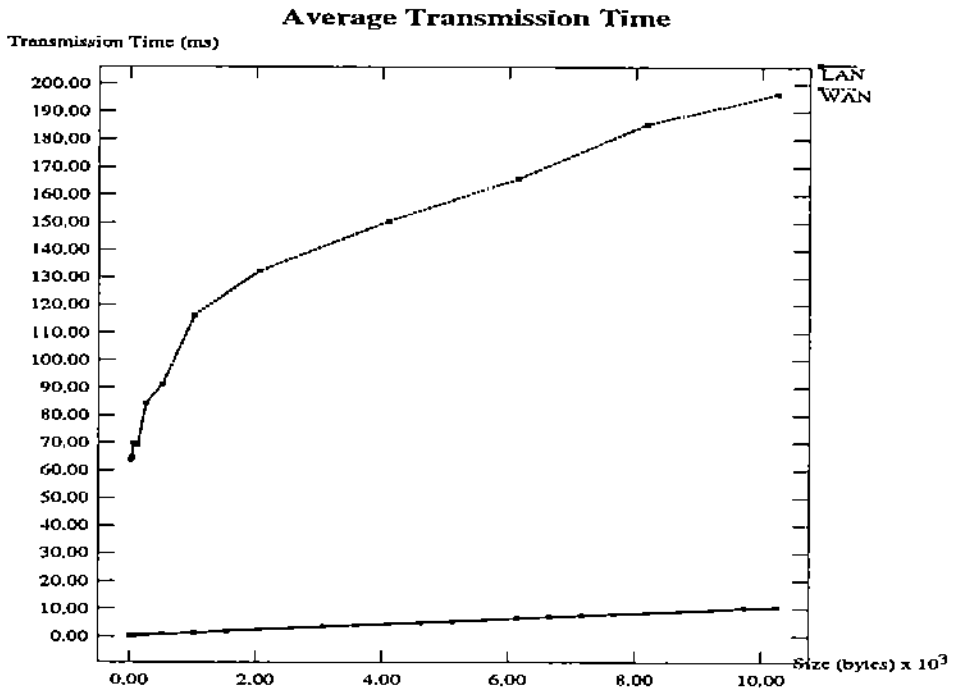

Figure 11: Transmission Time as a Function of Data Size

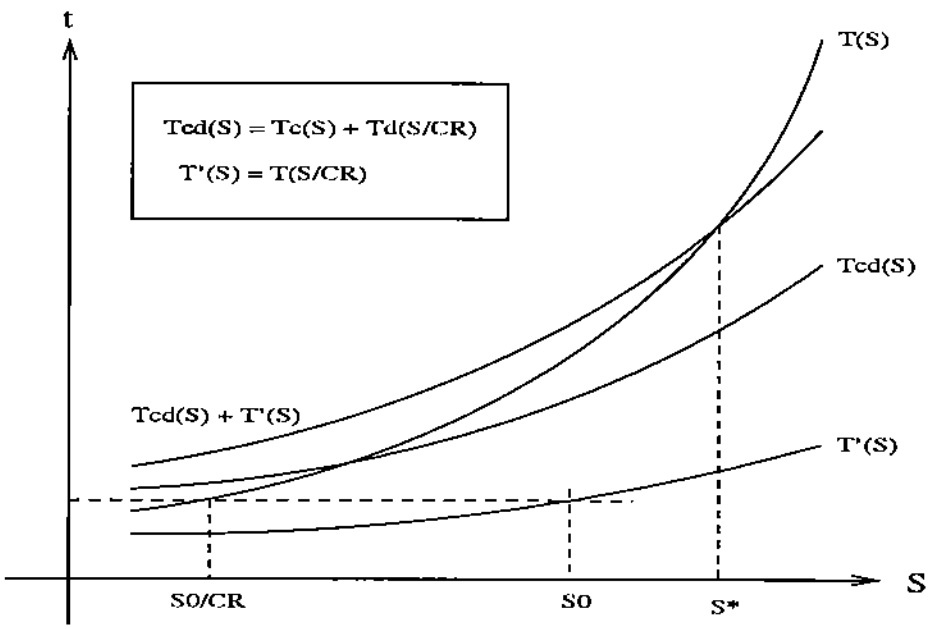

Figure 12: Data Size Threshold for Determining When Compression Is Needed

Table 2: PQ as a Function of CQ/DQ with Default HT

DQ \ CQ	10	20	30	40	50	60	70	80	90	100
10	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1
30	1	1	1	2	2	2	2	2	2	2
40	1	1	2	2	2	2	2	2	2	2
50	2	2	3	3	3	3	3	3	3	3
60	2	2	3	3	3	3	3	3	3	3
70	3	3	3	3	4	4	4	4	4	4
80	3	3	3	4	4	4	4	4	4	4
90	3	3	4	4	4	4	4	4	4	4
100	3	3	4	4	4	4	4	4	4	4

sion and decompression, and vice-versa. The matter is to find the threshold for the data size in order to be able to determine when compression is needed. Figure 12 illustrates this scenario. The threshold S^* will have to be determined by experiments.

Table 3: PQ as a Function of CQ/DQ with Optimal HT

DQ \ CQ	10	20	30	40	50	60	70	80	90	100
10	1	2	2	2	2	2	2	2	3	3
20	3	3	3	3	3	3	3	3	3	4
30	3	3	3	3	3	3	3	3	3	4
40	3	3	3	3	4	4	4	4	4	4
50	3	3	3	4	4	4	4	4	4	4
60	3	3	4	4	4	4	4	4	4	4
70	3	3	4	4	4	4	4	4	4	4
80	3	4	4	4	4	4	4	4	4	4
90	3	4	4	4	4	4	4	4	4	4
100	4	4	4	4	4	4	4	4	4	4

3.3.7 Presentation Quality as a Function of Compression Quality and Decompression Quality

Although CQ and DQ (as well as other factors) affect the presentation quality of an image, they are not a real measure of the image quality presented to users. Presentation Quality,

or PQ, is the image quality users perceive. It is subjective, and sometimes called *Perceptual Quality* ([11]).

The fact that PQ is affected by a combination of CQ, DQ, and the way HT is used (either *default* or *optimal*) can be expressed by a function, i.e., PQ = $\mathcal{Q}$(CQ, DQ, HT). Table 2 and table 3 indicate $\mathcal{Q}$(CQ, DQ, *default*) and $\mathcal{Q}$(CQ, DQ, *optimal*), respectively.

In both tables, PQ is presented using an integer indicating the relative levels of subjective presentation quality. 1, 2, 3, and 4 represent the levels of subjective presentation quality to be moderate, good, excellent, and indistinguishable from the original image, respectively. Clearly, function $\mathcal{Q}$ is a monotonically non-decreasing function of both CQ and DQ.

4 Quality of Service Test and Negotiation

4.1 Formulation of the QOS Test Problem

In last section, we established the (*numerical*) relationships between JPEG parameters and QOS parameters by evaluating the experimental data about a JPEG compressor and decompressor. They can be summarized as a set of n-ary non-linear equations:

$$\begin{cases} CR &= \mathcal{F}(CQ, S_0) \\ S_1 &= \mathcal{G}(CQ, S_0) \\ T_c &= T_c(CQ, S_0, N_f, HT) \\ T_d &= T_d(DQ, S_1, N_f, HT) \\ PQ &= \mathcal{Q}(CQ, DQ, HT) \\ T_t &= T_t(S) \\ T_s &= T_c + T_d + T_t \end{cases} \tag{7}$$

How can we use these equations to conduct QOS test as well as QOS negotiation if a QOS test fails? Remember that in a QOS specification, QOS parameters and their constraints are given as inputs. Given below is an example for a QOS specification for a MPEG-encoded video stream (table 4):

Our tasks are to find if there exist a set of quantities of JPEG parameters such that the given QOS specification can be satisfied, and if so, what these quantities are. To find a systematic approach to solving this problem, we formulate the problem in a formal way.

The problem of QOS test can be formulated as a multiple-objective, non-linear programming problem. Let $\mathbf{X} = (x_1, x_2, \ldots, x_n)^T$ be a vector for codec parameters (input parameters). Let $\mathbf{Y} = (y_1, y_2, \ldots, y_m)^T$ be a vector for QOS parameters (output parameters). The relationships between input parameters and output parameters can be described as the following functions:

$$y_i = f_i(x_1, x_2, \ldots, x_n) = f_i(\mathbf{X}) \qquad i = 1, 2, \ldots, m \tag{8}$$

Table 4: An Example of A QOS Specification

resolution	=	320 x 240,	negotiable.
color	=	8 bits,	negotiable.
frame_rate	=	20 fps,	negotiable, 15 fps minimum.
delay	=	50 ms,	non-tolerable.
jitter	=	20 ms,	non-tolerable.
loss_rate	$\leq$	1%,	tolerable up to 2%.
quality	$\geq$	3,	tolerable, no worse than (level) 2.

with respect to following constraints:

$$y_i^a \leq y_i \leq y_i^b \qquad i = 1, 2, \ldots, m \tag{9}$$

where $f_i (i = 1, 2, \ldots, m)$ are in general n-ary non-linear functions, and y_i^a and y_i^b are lower and upper bounds of y_i, respectively. Let $\mathbf{F(X)} = (f_1(X), f_2(X), \ldots, f_n(X))^T$, $Y^a = (y_1^a, y^a 2, \ldots, y_n^a)^T$, and $Y^b = (y_1^b, y^b 2, \ldots, y_n^b)^T$. Then equations (8) and (9) can be written as follows:

$$\mathbf{Y} = \mathbf{F(X)} \tag{10}$$

and

$$\mathbf{Y^a} \leq \mathbf{Y} \leq \mathbf{Y^b} \tag{11}$$

Clearly, a QOS specification can be represented as a set of constraints associated with the set of non-linear equations. Negotiating QOS means changing the constraints associated with the equations. The equations themselves don't change. The constraints form a subspace in an m-dimension space, with each valid vector **Y** being a point in this subspace. Finding a solution to the problem is equivalent to finding a vector **X** such that the corresponding vector **Y** falls into this subspace.

This is a multiple-objective, non-linear programming problem. There may be multiple solutions to it. However, getting all the solutions requires a great deal of computation. Fortunately, we don't need all the solutions; we need only one solution that is sufficiently good for the QOS specification. Although many well-known iterative approaches exist ([1] [3]), they are not most appropriate because of the following reasons:

- The functions f_i (i = 1, 2, ..., m) may not have mathematically closed forms. Therefore, they may not be differentiatable, as required by most traditional algorithms.
- The ranges of input parameters are discrete whereas in traditional algorithms, the ranges of input parameters are assumed to be continuous.

- The traditional approaches usually seek *optimal* solutions, whereas in our case, optimal solutions are not necessary.

Nevertheless, many of their ideas are applicable to our problem. For example, usually the multiple-objective problem is converted to single-objective problem by mapping in some way the multiple objectives to a single objective. Iterative methods are frequently used in solving non-linear programming problem. Below, we present an iterative algorithm to solve the set of equations so that one of its solutions, if any, suggests the desired setting of codec parameters.

4.2 An Algorithm for QOS Test and Negotiation

Since many functions do not have closed forms mathematically, numerical tables are established to approximate them. In our algorithm, we store discrete values in a table for each function. Values not directly stored in the tables are interpolated on-the-fly, according to their adjacent function values. Therefore, computing a function value simply means a table lookup. Tables are loaded into memory for use when the algorithm starts.

Before we present our algorithm, we have to understand one more fact in multiple-objective programming, that is, the optimal value for each objective (y_i) may not be reached at the same time. Furthermore, there is no general criterion to determine which one is better. In reality, we can sort y_i based on their relative importance and try to satisfy the most important objective first. QOS negotiation process can take advantage of this order information to speed up the process of finding the solutions.

Algorithm:

1. Set i = 0; choose an initial vector X_0.
2. Compute $Y_i = \mathbf{F}(X_i)$.
3. Check if the constraints $Y^a \leq Y_i \leq Y^b$ hold. If yes, stop, and X_i is one solution; if not, continue.
4. Set i = i + 1; set $X_i = X_{i-1} + \Delta_i$. If X_i has traversed all the search space, the algorithm terminates with no solution. Otherwise, go to step 2.

where Δ_i is an n-ary vector and should be chosen such that each component of X_i is incremented by some fraction, one at a time. This will become clear in the example below. There are several possibilities as to the results of this algorithm:

1. If there exists exact one solution, then clearly the QOS test succeeds with the codec-parameters set to values in the solution;
2. If there are multiple solutions, an optional search process starts, with a hope to find an optimal solution to the QOS specification. The optimality is meant by conforming some principles, or by in accordance with some policies, all to be described in the next subsection; and

3. If there is no solution, then the QOS test fails. To see if it is a conditional failure, all of the QOS parameters must be examined. If all of the QOS parameters are hard (unnegotiable), then the QOS test really fails and a report will be returned to the user, stating that the QOS specification is not realistic. If some of the parameters are soft (negotiable), then it is a conditional failure and a QOS negotiation may take place. The user may provide a new QOS specification, or the system may go ahead and make some adjustments to those soft parameters so that the new QOS test may be successful.

Example: *Given a QOS specification with two parameters: PQ and T_s. The constraints are that $PQ \geq \omega$ and $T_s \leq \tau$. The relationships between QOS parameters and JPEG parameters are given in equation (7). Use the algorithm to conduct QOS test. Assume the frame size and/or resolution are fixed and known. $N_f = 1$.*

1. Check if the QOS specification contains some unrealistic values. If so, the algorithm rejects the QOS specification; otherwise it continues. For example, if τ is so small that the end-to-end delay for a single packet is greater than it, then the constraint $T_s \leq \tau$ is not satisfiable, that is, there is no way to transmit any image over the network within τ.
2. Check the condition $T_t(S_0) \leq \tau$. If it is true, it means the QOS specification can be satisfied without compressing the original frame. Note that an uncompressed frame always provides highest quality possible, therefore, $PQ \geq \omega$ should be satisfied automatically. The algorithm continues if the condition is false, which implies compression is necessary to satisfy the QOS specification.
3. Set HT = *default*, i = 0.
4. Choose an initial value of $CQ = CQ^i$.
5. choose those DQ^i such that $Q(CQ^i, DQ^i, default) \geq \omega$ holds. For each DQ^i in the list, do step 6 and step 7. If the list is empty, set i = i + 1; increase the new value of CQ by a certain quantity. For example, $CQ^i = (CQ^{i-1} + CQ_{max})/2$. If $CQ^i \geq CQ_{max}$, go to step 9; otherwise, go to step 5.
6. Compute CR^i, S_1^i, T_c^i, T_d^i, T_t^i, according to equations (7).
7. Check the constraint $T_s^i \leq \tau$. If it is true, the algorithm stops with one solution: (CQ^i, DQ^i, HT). Otherwise, it continues.
8. Set i = i + 1; decrease the new value of CQ by certain quantity. For example, we can set $CQ^i = (CQ^{i-1} + CQ_{min})/2$. If $CQ^i < CQ_{min}$, go to step 9; otherwise go to step 5.
9. If HT was set to *default*, set HT = *optimal* and go to step 3; otherwise, the algorithm stops and rejects the QOS specification.

This algorithm is guaranteed to terminate in time proportional to $O(|DQ| \log |CQ|)$, where $|CQ| = CQ_{max} - CQ_{min}$ and $|DQ| = DQ_{max} - DQ_{min}$. We can reduce the running time to $O(\log |CQ| \log |DQ|)$ by using binary search on variable DQ.

4.3 Principles for QOS Negotiation

QOS negotiation is a very complicated task because it involves so many arguments. During a QOS negotiation process, there are usually many solutions to meeting a new QOS specification and there may not exist a best solution which can outperform others. Rather than comparing all the solutions and selecting the most appropriate one after finding all of them, it is necessary to develop a set of policies (principles or rules) to guide the QOS negotiation process so that only one solution that is most appropriate (under the policy used) will be found. We propose several principles below with a hope that some of them may get further development as useful policies in future practice.

Local Adjustment First Principle Some resources belong to the local host; others are shared or used by more than one machines. Choose the QOS parameters first that only affect the local host and won't lead to the resource reallocation at a global level. This will minimize a global effect in terms of resource allocation. For example, local buffer size adjustment doesn't necessarily lead to global resource re-allocation. This idea can be extended to the clusters of hosts.

Minimal Scope of Change Principle Choose those QOS parameters first that would cause the scope of changes to be minimum. Or choose the set of QOS parameters that is minimal in terms of the number of variables/parameters being changed.

Easiest Negotiation Principle Many QOS parameters are interrelated with each other. For example, compression quality is directly or indirectly related to many QOS parameters such as compression time, compression ratio, transmission time. Choose those QOS parameters first that have least dependency relationship with other QOS parameters. With the dependency graph, it is not difficult to find such parameters.

Fastest Negotiation Principle (Peer-to-peer) Negotiation may require several rounds of interactions among involving parties. It may also involve reconfiguration of running processes. Choose those QOS parameters first that would respond the fastest to the QOS negotiation requests. It is perceived to be useful in a fast changing environment.

Optimality Principle This principle leads to the development of different policies: least resources policy, best quality policy, shortest response time policy, and quality/response time pair policy, etc. Choose QOS parameters to obey these policies.

Maximality Principle Not all QOS specifications can be met even through re-negotiation. We can only seek to satisfy a subset of QOS parameters. Choose those QOS parameters that will lead to the satisfaction of the most QOS parameters/requirements. Of course, different QOS parameters may have different significances. We can prioritize them by assigning them different weights. The criterion may be the maximal weighted number of QOS parameters.

Equivalence Principle Define equivalent trade-off based on certain measures – in terms of resources, users' perception and response time, etc. Try to trade QOS parameters for others using this principle.

It is reasonable to obey these principles because each deals with some facets of the problem of QOS tradeoff. Which is the most appropriate still needs to be found out through experimentations. If none is superior to others, we need then to find out under which conditions one principle outperforms others, or which principles perform better for certain applications than others.

4.4 Discussions

Our algorithm can be applied to many applications, regardless of whether the data is live or pre-orchestrated and stored in secondary or tertiary storage. Equation

$$T_s = T_c + T_d + T_t$$

works in video-conferencing-type of real time applications in which data are live and compressed on-the-fly. In video-on-demand- or video database-type of applications, this equation will be replaced with:

$$T_s = T_d + T_t$$

Here I/O time (e.g., time spent in loading data to memory) has been ignored because it is an invariant. Note that in MPEG-encoded video applications, some functions in equations (7) need to be re-examined. For instance, unlike in MJPEG where the average compressed frame size is almost a constant (equation 5), the average MPEG-compressed frame size fluctuates, depending upon whether the frame type is I, P, or B. However, similar pattern can be observed for group of pictures (GOP). Here we refer to a GOP as a frame sequence both starting and ending with two consecutive I-frames. Therefore, with a little bit modification of the functions/equations involved, the algorithm remains applicable. In image database or digital library, only still images are involved. Furthermore, they may be pre-compressed with different codec schemes and stored with various levels of resolutions. In this case, N_f can be set to 1. The codec scheme, along with the resolution level, serves as the role of CQ in JPEG. The set of equations may be different, but the algorithm remains usable.

Figure 13 depicts how a QOS test is involved in a setup phase for a distributed multimedia applications. Note, however, that setup phase of a connection is not the only place where QOS tests are involved. QOS tests are also involved during the course of communication services. As soon as the execution condition changes which affects the resource utilization, QOS may need to be re-negotiated to accommodate the new environment. QOS tests will be conducted again, leading to re-allocation of system resources. Codec-related parameters may also need to be adjusted from time to time. Of course, these adjustments can be made adaptive, too.

There are some limitations in our formulation of the problem. Basically, the factors that influence QOS tests can be categorized into two classes: static and/or statistical information

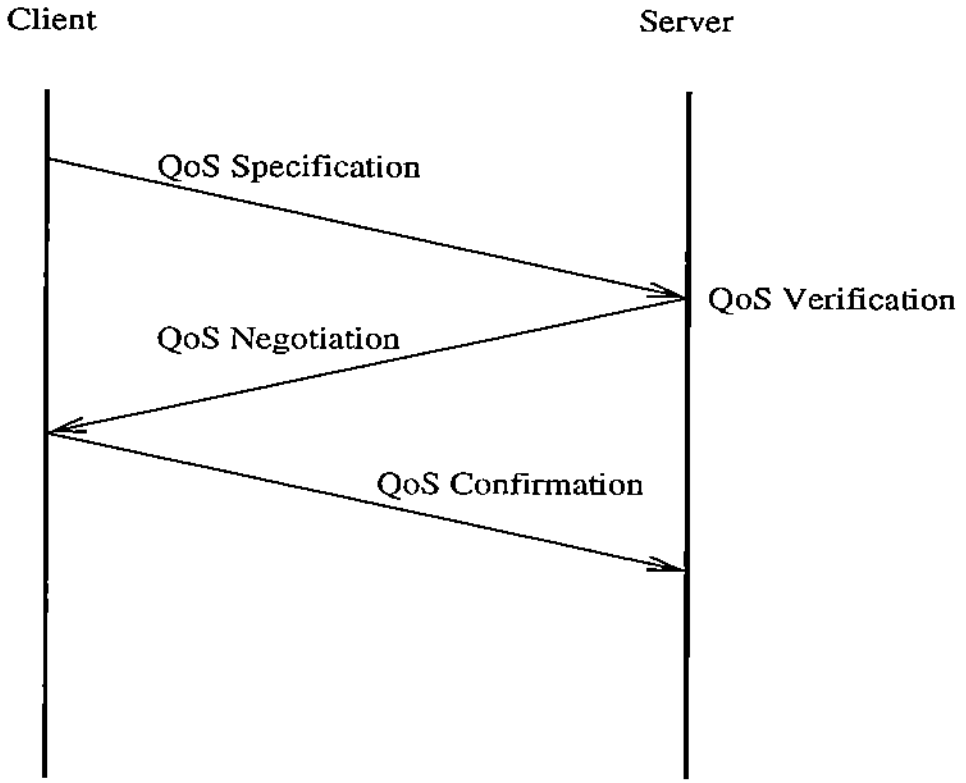

Figure 13: Setup Phase of A Client-Server Connection for Video Applications

about each component of the system and dynamic and/or run-time behaviors of the system. The examples of the former are average network latency to participating remote sites, average machine load, the link capacity of the underlying network, the MIPS of the host, memory, disk volume, etc. The statistical performance information about a codec scheme falls into this category, too. The latter includes currently available network bandwidth, current machine load, end-to-end delay among involving sites, etc. Our formulation does not model the system factors (e.g., resources availability and utilization, system computation power) other than the application codec module, nor it model the dynamic nature of the system.

Part of the reasons for excluding these factors from consideration in our formulation is the complexity of the problem as well as lack of modeling tools for doing so. This is a big challenge for systematically studying the problem of QOS tests and QOS negotiation for a better QOS control in a distributed environment.

The fact that these system-related factors are not formulated in our problem doesn't necessarily prevent us from considering them in our algorithm, however. We can collect and utilize some of the information in our algorithm. For example, at the setup time of a connection, when a user submits her request for service, she can also report a statistical information about her local site (machine, network bandwidth) and embed it into the requesting packet. She can

also record a timestamp on the packet so that when the server receives this request, it can estimate the round trip time. Moreover, the distance (or the number of hops) between the client and the server as well as their end-to-end delay, which the value of T_s highly depends on, can be collected in the similar way. The server will know the number of hops between it and the client as soon as it receives the very first requesting packet from the client. An approximate measure of the end-to-end delay will also be available to the server. The server can then use these information in QOS tests, thus narrowing down the search space and helping get more accurate solutions.

4.5 Self-Adaptive Huffman Table Computation for MJPEG-based Video Applications

By now it should be clear that properly setting JPEG parameters is critical to QOS test and the performance of distributed video applications. The setting of these parameters should be based not just on one figure or two, but rather on the integration (or synthesis) of all figures. For example, CQ, CR, T_s, and PQ are all related.

Because of the existence of so many alternatives, QOS control should be performed in a policy-based fashion. Policies can be made according to application requirements. For example, an application can specify that presentation quality is more important than data size and/or response time. Then QOS control can be performed to conform this policy by setting the parameters to some values that would guarantee high presentation quality.

Policy-based QOS control must be combined with adaptation approach to better adjust the system's QOS to current running environments. Various levels of adaptability can be explored in distributed multimedia systems. At system and network level, adaptive error recovery scheme must be designed in the transport protocol to accommodate various physical environments. At the application level, application-specific adaptability features are extremely useful in meeting QOS requirements. ([10]). Adaptability also comes into play within a specific codec scheme by dynamically adjusting some codec scheme-specific parameters. Adaptability features of an application, including those related to codec schemes must be explored. Below, we will propose a self-adaptive JPEG scheme that explores the adaptability of Huffman Table computation.

In MJPEG, we can adaptively compute the Huffman Table used in the phase of entropy coding. There are traditionally two approaches to compute and use the Huffman Table. One is static, that is, to precompute a Huffman Table for the first frame and fix it for all the succeeding frames, or alternatively to load an ISO-defined Huffman Table for all the upcoming frames. Either way, the Huffman Table is computed/loaded and reliably transmitted only once, so it takes less time to compute/load, compress, and transmit. However, the resulting compressed image sequence may not be optimal. The other approach is dynamic, that is, to compute the Huffman Table on-the-fly. One table is optimally computed and transmitted for

each image. So it takes more time to compute, compress, and transmit, but the compressed image sequence (excluding the table themselves) is optimal. As the experiments indicated, for some QOS parameters, dynamic approach performs better; for others, static approach performs better.

We can come up with a generic approach so that the above two approaches become its special cases. The generic approach computes and transmits a Huffman Table every N consecutive frames (or GOPs in general) in a video sequence. Clearly, the static approach corresponds to the case of $N = 1$, and the dynamic approach to the case of $N = \infty$, respectively. In general, N is a parameter and can be set either statically or dynamically. To achieve adaptability, N must be tuned dynamically, based upon some run time criteria, such as whether the amount of scene changes exceeds a threshold. This way, the computation and transmission of a new Huffman Table is triggered whenever the amount of scene changes exceeds a threshold. Figure 14 shows the algorithm for adaptively computing Huffman Tables using the criterion mentioned above.

```
Algorithm Adaptive_Huffman_Table
begin
   I'= 0
   while ((I = get_image from input device) != NULL)
   begin
      Compress the image I
      if |I - I'| > threshold
      then
        Compute Huffman Table
        Transmit it along with the compressed image
      else
        Transmit the compressed image
      endif
      I' = I
   end
end
```

Figure 14: Algorithm for Adaptively Computing Huffman Tables

With this approach, we can achieve a good balance between the computation cost and the transmission and storage cost.

5 Conclusion

A QOS test is the first step toward QOS control in a distributed real-time multimedia application. Its result highly relies upon the performance of the codec involved. In this paper, we have conducted a series of experiments, evaluated the performance of a commercial implementation of JPEG baseline sequential compressor, and used it as a vehicle in QOS tests. We have formulated the problem of QOS tests as a multiple-objective, non-linear programming problem and presented an algorithm to solve it. General principles for QOS negotiation have been proposed. The limitation of our formulation and further considerations for improving it have been discussed. Finally, an adaptive scheme for Huffman Table computation in distributed video applications is proposed.

References

[1] C. L. Hwang and A. S. Md. Masud. *Multiple Objective Decision Making, Methods and Applications: A State-of-the-Art Survey.* Springer-Verlag, Berlin, 1979.

[2] Craig Partridge. *Gigabit Networking.* Addison-Wesley, 1994.

[3] David G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 1984.

[4] Willibald A. Doeringer, Doug Dykeman, Matthias Kaiserswerth, Bernd Werner Meister, and Harry Rudin. A survey of light-weight transport protocols for high-speed network. *IEEE Transactions on Communications*, 38(11):2025–2039, 1990.

[5] Sylvie Dupuy, Wassim Tawbi, and Eric Horlait. Protocols for high speed multimedia communications networks. *Computer Communications*, 15(6):349–358, 1992.

[6] D. Ferrari. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(4):368–379, 1990.

[7] D. Ferrari. Distributed Delay Jitter Control in Packet-Switching Internetwork. *Internetworking: Research and Experience*, 4:1–20, March 1993.

[8] Didier Gall. MPEG: A Video Compression Standard for Multimedia Applications. *The Communication of the ACM*, 34(10), Oct. 1991.

[9] Shunge Li. Quality of Service (QOS) Measurement and Control for Distributed Multimedia Systems. Ph.D Thesis Preliminary Proposal, Department of Computer Sciences, Purdue University, Apr. 1996.

[10] Shunge Li, Shalab Goel, and Bharat Bhargava. VC Collaborator: A Mechanism for Video Conferencing Support. In *SPIE Photonics East '95 Symposium – First International Symposium on Photonics Technologies and Systems for Voice, Video, and Data Communications, SPIE Proceedings Vol. 2617*, pages 89–99, Philadelphia, Pennsylvania U.S.A, October 1995.

[11] Klara Nahrstedt and Lintian Qiao. A Tunning System for Distributed Multimedia Applications. Technical Report UIUCDCS-R-96-1958, University of Illinois at Urbana-Champaign, Department of Computer Science, July 1996.

[12] Klara Nahrstedt and Ralf Steinmetz. Resource Management in Networked Multimedia Systems. *IEEE Computer*, 28(5):52–63, May 1995.

[13] Doug Shepherd, David Hutchinson, Francisco Garcia, and Geoff Coulson. Protocol support for distributed multimedia applications. *Computer Communications*, 15(6):359–366, 1992.

[14] Brian C. Smith. *Implementation Techniques for Continuous Media Systems and Applications.* PhD thesis, Department of EECS, Computer Science Division, University of California at Berkeley, 1994.

[15] Sun Microsystems, Inc. *Solaris XIL 1.1 Imaging Library Programmer's Guide.* November 1993.

[16] Andreas Vogel, Brigitte Kerherve, Gregor von Bochmann, and Jan Gecsei. Distributed Multimedia and QOS: A Survey. *IEEE Multimedia*, 2(2):10–19, Summer 1995.

[17] Carsten Vogt. Quality-of-Service Management for Multimedia Streams with Fixed Arrival Periods and Variable Frame Sizes. *Multimedia Systems*, 3(2):66–75, May, 1995.

[18] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *The Communication of the ACM*, 34(10), Oct. 1991.

[19] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 9(5), September, 1993.