University of Alberta

Wavelet Video Coding with Application in Network Streaming

by

Yufei Yuan

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2005

Abstract

In last decade, Internet broadcasting has experienced explosive growth along with the Internet infrastructure. Most current Internet video applications involve web-based video playback where pre-compressed video is streamed from a server to a client upon request. Many video compression schemes optimize their reconstruction quality for a specific target bitrate. However, a fixed transmission rate is not suitable for Internet video transmission, because the communication channel is shared among users and the bandwidth available to a specific user is time-varying. In this dissertation, we propose a solution to this problem by adopting a joint source-channel coding (JSCC) approach to design a wavelet-based fine granular scalable (WavFiGS) video codec. The WavFiGS coding system consists of a base layer (BL) codec that employs a hybrid coding – motion compensation with discrete wavelet transform (DWT) – architecture and an error-resilient enhancement layer (EL) codec that uses a robust progressive bit-plane coding scheme.

The research work starts with the design of an efficient wavelet-based still image codec. The objective is to design with simple and efficient techniques that achieves good compression performance and maintains a low computational complexity. We have demonstrated with our proposed schemes that a relatively simple context modeling technique, when coupled with other well-known techniques, achieves better compression than other more complex schemes.

The simplicity of the proposed schemes stimulated our efforts to exploit their potential. We discovered that the information asymmetry between an encoder and a decoder could be exploited so that all bit-planes within a pyramid can be encoded simultaneously. This is a major breakthrough in parallel wavelet image coding since it provides a new dimension of processing flexibility in addition to the conventional

parallelization approaches. It was also discovered that in the embedded image coding, the encoding order of refinement bits within one bit-plane has little or no impact on the peak signal-to-noise ratio (PSNR) quality of the reconstructed image. This concept of parallel bit-plane coding is general so that parallelization of other wavelet image coding schemes could also benefit from it.

The proposed still image coding scheme is employed in the intra-frame coding of the BL. The inter-frame residual coding is completed by the proposed macro-block stack-run codec. The EL codec is based on error-resilient elementary Golomb (EG) coding and progressive bit-plane coding. The output bitstream of the EL coder can be fragmented in such a way that each delivered EL packet is decoded independently and contributes to the improvement of the reconstruction quality of the decoded video.

Finally, we propose an efficient transmission strategy to stream an FGS video. Based on the justification that the BL is essential to the reconstruction of a video signal and that the delivery of the EL is less critical, it is proposed that the BL be protected by forward error correction (FEC). The EL be left without any FEC and is only protected by the error resilience incorporated into the EL source coder. We also establish an analytical model to optimally determine the channel coding rate under different packet loss probabilities. The effectiveness of both the streaming strategy and the analytical model was corroborated by our experimental results.

Acknowledgements

The work presented in this thesis would not have been possible without the free and open source software communities. Kudos to the many developers and researchers, too many to mention individually, for their fantastic work in the GNU/Linux operating system, development tools, software libraries, the NS-2 network simulator, and the LaTeX word processor, etc.

I am grateful to Prof. Anup Basu, Prof. Bruce Cockburn, and Prof. Zoltan Koles for serving on my advisory committee and for many insightful discussions and suggestions that have greatly improved the quality of the research work and the thesis writing. I am also thankful to Prof. Amir Asif of York Univeristy for serving as the external examiner.

Over the course of my graduate career, I benefited invaluably from my thesis advisor, Mrinal Mandal. His constant encouragement, consistent enthusiasm, technical and theoretic savvy, and generosity with his time all combine to make him a model research advisor. I strive to emulate his ability to strike a balance between rigorous theoretical foundation and engineering feasibility. I especially thank Mrinal for "showing me the ropes" of academia and for treating me as a colleague.

Lastly, but most important, I thank my family – my parents, siblings, and especially my wife Yuqing – who together form the caring and supportive environment that is the sole reason for any progress I might have made. Above all, I am thankful to Yuqing's love, support, and confidence, which is in the end, all that matters.

Contents

Contents ii

List of Figures vi

List of Tables xi

1 Introduction 1

1.1 Motivation 5

1.2 Problem Statement 9

1.2.1 Network-Aware Wavelet FGS Video Coder 9

1.2.2 Streaming System 10

1.3 Thesis Overview 11

2 Related Work 13

2.1 Wavelet Image and Video Compression 13

2.1.1 Dyadic Wavelet Transform 13

2.1.2 Wavelet Image Coding 18

2.1.3 Wavelet Video Coding 21

2.2 Scalability in Video Coding 23

2.2.1 SNR Scalability 25

2.2.2 Temporal Scalability 25

2.2.3 Spatial Scalability 26

2.2.4 Fine Granularity Scalability 27
2.3 Error Resilience 28
2.3.1 Robust Entropy Coding 30
2.3.2 Multiple Description Coding 30
2.3.3 Layered Coding with Unequal Error Protection 31
2.4 Network Characteristics 32
2.5 Internet Video Streaming 36

3 Efficient Wavelet Based Still Image Coding 38
3.1 Wavelet Difference Reduction (WDR) Image Coding 39
3.2 Fast and PSNR-Improved WDR Algorithms 41
3.3 Context Modeling WDR 53
3.4 Color Image Coding with Context Modeling 62
3.4.1 Color Wavelet Difference Reduction 64
3.4.2 Experimental Results 67
3.5 Summary 69

4 Parallel Image Compression Through Bit-Plane Coding 71
4.1 Overview of Parallel Wavelet Image Coding 72
4.2 Parallelization Through Data-Independent Bit-Plane Encoding 73
4.2.1 Coefficient Reordering By Linear Indexing 73
4.2.2 Bit-Plane Partitioning and Run-Length Encoding 74
4.2.3 Parallel Bit-Plane Encoding 75
4.3 Bitstream Compatibility in Parallel Bit-Plane Encoding 79
4.4 Sequential Bit-Plane Decoding 83
4.5 Experimental Results 84
4.6 Summary 89

5 Wavelet Scalable Video Coding **92**

5.1 Spatial Scalable Video Coding with Hierarchical Backward Motion Compensation 93

5.1.1 Backward Motion Compensation 93

5.1.2 New Low-Band-Shifting-Based Scheme 95

5.2 Wavelet Fine Granular Scalable (WavFiGS) Video Coding 105

5.2.1 Macro-block Stack-Run Coding 106

5.2.1.1 Algorithmic Details of The MBSR 108

5.2.1.2 Experimental Results 111

5.2.2 Error Resilient Enhancement Layer Coding 116

5.2.2.1 Error Resilience from Localization and Decoupling . 118

5.2.2.2 Experimental Results 124

5.3 Summary 127

6 Forward Error Correction in Packetized Video Streaming **128**

6.1 FEC Performance in Video Streaming 128

6.1.1 Parameters and Variables 130

6.1.2 Playable Frame Rate 132

6.2 Non-Scalable Video Streaming Using FEC 136

6.2.1 Model-Based Analysis 136

6.2.2 Simulation-Based Analysis 144

6.3 FGS Video Streaming Using FEC 151

6.4 Summary 162

7 Conclusions **163**

7.1 Contributions 163

7.2 Future Work 166

7.3 Publications 168

Bibliography **170**

A The Design and Implementation of The Streaming System **186**

A.1 Streaming System for Non-Scalable Video 186

A.1.1 Streaming Server Module 187

A.1.2 Receiver Buffer Module 193

A.1.3 Video Decoder Module 194

A.2 Streaming System for FGS Video 195

A.2.1 FGS Video Streaming Server 195

A.2.2 FGS Video Client 198

List of Figures

1.1 Users can access a video database over networks for streaming video content. 2

1.2 Structure of a live video streaming system. 4

1.3 The priority hierarchy inside a typical output bitstream of a scalable video coder. 8

2.1 Multi-resolution representation of the *Lena* image using DWT. 14

2.2 One-dimensional dyadic wavelet transform implemented by a two-channel perfect reconstruction filter-bank. 14

2.3 A typical transform-based lossy image encoder. 18

2.4 (a) Zerotree data structure and (b) subband scanning order. 19

2.5 Generic DCT-based hybrid video encoder [6]. 24

2.6 SNR scalability decoder architecture defined in MPEG-2. 26

2.7 Performance of different video coding schemes. 28

2.8 FGS encoder architecture [59]. 29

2.9 The syntax of the RVLC word in the MPEG-4 standard 31

2.10 Generic block diagram for multiple description coding. D-X represents the X-th description of the coded data. 31

2.11 Protocol stacks for media streaming. 36

3.1 (a) Plot of Δl_{n+1} versus p and p_s. (b) An enlarged portion of (a), where $\Delta l_{n+1} > 0$. (c) The contour plot of $\Delta l_{n+1} = 0$ and $p = p_s$. It is shown that $\Delta l_{n+1} < 0$ when $p_s < p$. 47

3.2 An example two-level wavelet pyramid of size 4×4. The scanning order is illustrated by the corresponding index for each coefficient. . . 49

3.3 The comparison of timing performance on the coding of the *Lena* image (512×512, 8 bpp): (a) encoding of *Lena* at various bitrates; (b) decoding of *Lena* at various bitrates. 52

3.4 The composition of the output bit-stream. 64

3.5 Coefficient A in Y is modeled by its parent and neighbors whilst A's peer coefficients in U and V are directly modeled by A. 65

3.6 Prediction efficacy of the context-model used on *Girl* at a decomposition level of 6: (a) the significant coefficients (white pels) at the B_2 of U, (b) the significant coefficients in (a) that fail to be identified by context-modeling, (c) the significant coefficients at the B_2 of V, (b) the significant coefficients in (c) that fail to be identified by context-modeling. 66

3.7 The decoding output of the *Peppers* at compression ratio of 96:1. . . 70

4.1 Comparison between conventional parallelization strategy and the new strategy established in this paper. The new strategy at the bottom provides another dimension of processing flexibility for an MIMD architecture. 75

4.2 Processing architecture of the proposed bit-plane encoder, where B_n denotes the generated bitstream for the n-th bit-plane. 77

4.3 One example to illustrate the fixed scan-order and the bit-plane encoding. 79

4.4 The data structure that stores the order of the refining bits for the encoding of B_p. 81
4.5 The mean and the range of samples of the execution time of the SPIHT and the PESD-A on *Lena*. 86
4.6 Comparison of execution time of the SPIHT without AC and the PESD-A on *Lena*. 86
4.7 Comparison of execution time of the SPIHT without AC and the PESD-A on *Barbara*. 87
4.8 Comparison on visual quality of reconstructed *Lena* and *Barbara* when they are encoded by the PESD-A and the SPIHT at 0.15 bpp. 88
4.9 Comparison of execution time on a dual-processor server using the PESD-A, the PESD-B, and the PESD-B with bitstream compatibility using *Lena* and *Barbara*. 89

5.1 The two-level wavelet decomposition and the subband naming convention. 94
5.2 Block diagram for one-level of ME. H_0 is the low-pass analysis filter. L is the low-pass interpolation filter. Note that for simplicity, only one-dimensional filtering, decimation, and interpolation are shown. . . 95
5.3 Examples of the coefficients from the *Haar* wavelet for a 1-D signal $X_1[n]$ and its shifted-by-one-grid equivalent $X_2[n]$. H_0 is the low-pass analysis filter and G_0 is the low-pass synthesis filter. The amplitudes of the coefficients are not exactly as shown. Here, we are more interested in the shape of the output signal. 95
5.4 The building block of the proposed ME/MC scheme. C denotes combination of the four predicted subbands. 96
5.5 Quantized causal neighborhood of current coefficient. 102

5.6 Coding results on the *Football* SIF sequence. Comparison of the overall PSNR of the luminance (Y) components at 500 Kbps and 30 fps. . . 103
5.7 Coding results on the *Football* SIF sequence. Comparison of the overall PSNR of the luminance (Y) components at 500 Kbps and 30 fps. . . 104
5.8 The arrangement and indexing of blocks within one MB. 108
5.9 The inter-block scan-orders for the intra-coded MB and the inter-coded MB. 110
5.10 The dead-zone uniform quantizer used, where $T = q$. 111
5.11 Comparison between the H.263 and the MBSR on encoding 300 frames of the *Mobile & Calendar* CIF sequence at 200 Kbps and 10 fps. The PSNR values of the Y component are plotted. 113
5.12 The comparison on the PSNR quality of the four schemes. The 300 frames of the *Foreman* sequence are encoded at 350 Kbps and 30 fps. 116
5.13 The reconstructed 100th frame of the *Foreman* sequence in display order, which is encoded as a P-frame. 117
5.14 The scan order of the coefficients of macro-blocks within one slice. . . 121
5.15 The complete architecture of the WavFiGS encoder. 125
5.16 The average PSNR of Y component results obtained from encoding two sequences by the WavFiGS EL coder. 126

6.1 The structure of the m-th GoP and the inter-frame dependency relationship within it. 132
6.2 Comparison on ratio of PFR vs. source frame rate when $t_{RTT} = 50$ ms and $t_{RTO} = 200$ ms. 139
6.3 Comparison on the ratio of PFR vs. source frame rate when $t_{RTT} = 25$ ms and $t_{RTO} = 100$ ms. 140
6.4 The TCP-friendly transmission packet rate vs. the packet rate demanded by FEC-protected streaming applications. 141

6.5 Adjusting FEC configuration to search for the best one under specific network loss probability. 142
6.6 The effect of adjusting FEC configuration on the performance of video streaming using an MPEG-4 non-scalable source trace. 150
6.7 The PSNR of the reconstructed FGS frames at $p = 0.001$. The BL frames were protected with FEC and suffered no damage. 156
6.8 The PSNR difference of the two reconstructed FGS sequences at $p = 0.001$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity). 158
6.9 The PSNR difference of the two reconstructed FGS sequences at $p = 0.05$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity). 159
6.10 The PSNR difference of the two reconstructed FGS sequences at $p = 0.2$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity). 160
6.11 The PSNR difference of the two reconstructed FGS sequences at $p = 0.4$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity). 161

A.1 The generation of FEC redundant packets. 189
A.2 An example on the frame size of compressed video. 191
A.3 Packet scheduling for the transmission of groups of packets. 192
A.4 The ring buffer used to store the delivered EL data. Both pointers can only move clockwise. 199

List of Tables

3.1 The PSNR performance of the algorithms on the compression of *Lena*. 53

3.2 The PSNR performance of the algorithms on the compression of *Goldhill*. 54

3.3 The PSNR performance of the algorithms on the compression of *Boat*. 54

3.4 The PSNR performance of the algorithms on the compression of *Barbara*. 54

3.5 PSNR (dB) vs. bitrate (bpp) for SPIHT, SPIHT-AC, WDR, and CM-WDR. 61

3.6 Comparison on the number of encoded significant coefficients for the CM-WDR, the CM-WDR/AC, and the WDR. 62

3.7 Color SNR (dB) and component PSNR (dB) versus bitrate for CEZW, JPEG-2000, and CWDR on *Girls*. 68

3.8 Color SNR (dB) and component PSNR (dB) versus bitrate for JPEG-2000 and CWDR on *Lena*. 69

3.9 Color SNR (dB) and component PSNR (dB) versus bit rate for JPEG-2000 and CWDR on *Peppers*. 69

4.1 Comparison of PSNR values for the SPIHT, the SPIHT-AC [28], the MWDR, and the PESD. 90

5.1 Results for encoding 96 frames of the *Carphone* QCIF sequence at 30 fps using different wavelets. 112

5.2 The average PSNR results over entire sequences. Bitrate is in Kbps. . 114

5.3 Format of the WavFiGS EL source trace file. 127

6.1 Network settings. 137

6.2 The relative complexity when generating redundant packets with $s_I =$ 24, $s_P = 8$, and $s_B = 3$. 144

6.3 Format of the MPEG-4 source trace file. 145

6.4 The FEC configurations under different network loss probabilities. . . 155

A.1 Labels for decoded frames under all possible situations. 195

List of Algorithms

1 The CWDR encoder. 65

2 The bit-plane encoder. 78

3 The bitstream-compatible bit-plane encoder. 82

4 The pseudo-code for the bit-plane decoder. 84

5 Error resilient slice bit-plane encoder using elementary Golomb codes. 122

6 Slice bit-plane encoder without error resilience using elementary Golomb codes. 124

7 The pseudo-code for packing bit-planes for EL streaming. 197

List of Abbreviations

1-D	one-dimensional
2-D	two-dimensional
AC	arithmetic coding/coder
AIMD	additive increase multiplicative decrease
ALF	application layer framing
ARQ	automatic retransmission request
BL	base layer
BMA	block matching algorithm
BMC	backward motion compensation
bpp	bits per pixel
CBR	constant bit-rate
CIF	common intermediate format
DCT	discrete cosine transform
DWT	discrete wavelet transform
EG	elementary Golomb
EL	enhancement layer
EQ	estimation-quantization
EZW	embedded zerotree wavelet
FEC	forward error correction
FGS	fine granularity scalability/fine granular scalable

FIR	finite impulse response
fps	frames per second
GGD	generalized Gaussian distribution
GoP	group of pictures
HBMC	hierarchical backward motion compensation
HDTV	high definition television
ICS	insignificant coefficients set
IDCT	inverse discrete cosine transform
IP	Internet protocol
ISDN	Integrated Services Digital Network
JPEG	Joint Picture Experts Group
JSCC	joint source channel coding
Kbps	kilobits per second
LAN	local area network
LBS	low band shifting
MB	macro-block
Mbps	megabits per second
MBSR	macro-block stack-run
MC	motion compensation
MDC	multiple description coding
ME	motion estimation
MIMD	multiple instructions multiple data
MPEG	Motion Picture Experts Group
MSB	most significant bit
MTU	maximum transmission unit
PE	processing element
PFR	playable frame rate

PSNR	peak signal-to-noise ratio
PSTN	program-switched telephone network
OBMC	overlapped block motion compensation
QCIF	quarter common intermediate format
QoS	quality of service
RLC	run-length coding/coder
RS	run sub-array/Reed-Solomon
RTCP	real-time transmission control protocol
RTP	real-time transmission protocol
RTSP	real-time streaming protocol
RTT	round-trip time
RVLC	reversible variable-length code
SCS	significant coefficients set
SDP	session description protocol
SIF	storage intermediate format
SIP	session initiation protocol
SNR	signal-to-noise ratio
SNS	significant neighbor sub-array
SPIHT	set partitioning in hierarchical trees
SPS	significant parent sub-array
SR	stack-run
TCP	transmission control protocol
TPS	temporary set
UDP	user datagram protocol
UEP	unequal error protection
VCD	video compact disc
VLC	variable length coding/code

VLD	variable length decoding
VoD	video on demand
WDR	wavelet difference reduction

Chapter 1

Introduction

Digital video applications have become very popular over the last decade. In East Asian countries, the number of families that own at least one video compact disc (VCD) player has exceeded the number of families that own a video cassette recorder for many years [11]. Video-on-demand (VoD) has also become a service that is frequently offered by hotels and cable companies. Fig. 1.1 shows a scene that is familiar to many of us, in which the pre-stored compressed video is streamed over the Internet (or cable network) to customers. The interest in research and development for video streaming services over lossy packet networks, especially the Internet, has been aroused in recent years. The demand for streaming video has increased significantly [65]. An important driving force behind this momentum comes from the network equipment manufacturers and the network operation companies.

It has been long argued that the bottleneck of the so-called information highway lies at the "last mile", which means that the end-to-end bandwidth a user observes is limited by the connection bandwidth to her Internet service provider (ISP). However, it turns out that most Internet users are fairly satisfied with the services they are currently using and simply have no urge to migrate to broadband access [87]. For services such as e-mail, web browsing, and audio streaming, a dial-up link at 56

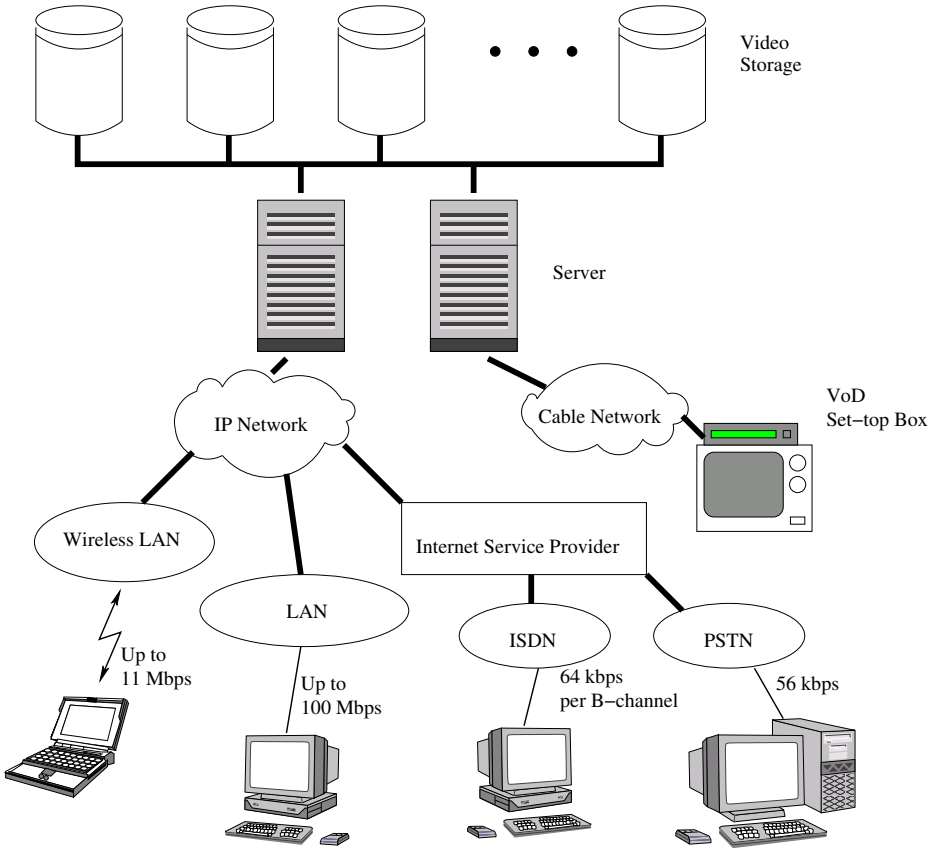


Figure 1.1: Users can access a video database over networks for streaming video content.

Kbps satisfies most people. But 56 Kbps is barely enough for even the lowest-quality streamed video. For the ISPs, services involving visual and graphical applications, such as VoD and on-line video games, are expected to be the next "big thing" that will encourage users to subscribe to broadband services and bring in new revenues [24].

Since the Internet is a heterogeneous network and supports real-time services only in a "best effort" manner, it is not an easy task to design high quality video streaming systems that are capable of coping with the Internet's unpredictable and time-varying network conditions as well as end users' divergent connection bandwidths to the Internet. The degradation in the visual quality of streamed video is mainly determined by the packet loss characteristics observed at the receiver and the inherent characteristics of the compressed bitstream generated by the video encoder.

The problem of transmitting video over lossy packet networks has essentially two main components: video compression and transmission strategy, e.g., communication protocols, as illustrated in Fig. 1.2. Traditionally, Shannon's source-channel separation theorem is used to design network video communication systems [35]. In Shannon's framework, an information source produces a string of symbols from some fixed alphabet and the *source coding* problem is to encode the source symbols in a series of codewords that minimizes the average codeword length (based on the given probability distribution of the input symbols). In turn, the codewords are transmitted across a noisy communication channel that corrupts the codewords according to a certain probabilistic model. Shannon showed that the probability of receiving a symbol incorrectly could be made arbitrarily small by introducing a *channel coding* stage [35] provided that the transmission rate is not greater than the channel capacity. By appending redundant bits to each code word or to collections of codewords, transmission bit errors can be detected and corrected. Shannon showed that the two problems of compression and error correction could be separated without sacrificing optimality provided that we can process an arbitrarily large amount of data and are willing to incur unbounded coding delays [35].

From the source design point-of-view, the video encoder is treated as a typical source coder. A popular approach consists of designing a video coder achieving a high compression ratio, optionally protecting an output bitstream with channel codes, and using one of the standard Internet transport protocols (TCP, UDP, etc.) to transmit the bitstream. The main drawback of this approach is that the system is designed to be forward only without feedback. The system can choose to use channel codes to protect the source bitstream, but it does not know when to use channel codes and how to choose parameters for channel coding because information on time-varying channel conditions is not available to the encoder (sender). From the channel design point-of-view, the lossy packet networks are seen as error-prone channels. Hence, researchers

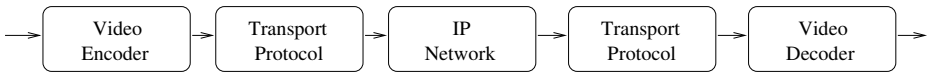


Figure 1.2: Structure of a live video streaming system.

have designed new transport protocols according to channel models without trying to optimize video source coding algorithms for the channels. The main drawback of this approach is that the overall video quality is limited by the nature of the video source coder used. Since no protocol can ensure error-free transmission under real-time delay constraints, the quality of the reconstructed video is inferior to that of error-resilient video coders.

Because video communication is under a strict real-time delay constraint, many of the latest video communication systems adopt a design strategy known as joint source channel coding (JSCC). Here, the design of a video source coder is done jointly with the transmission protocols (as in Internet) or channel characteristics (as in wireless/satellite channels). To design a high-performance JSCC video communication system for streaming video over lossy packet networks, there are essentially four issues that need to be considered. First, we need to design a scalable video coder that performs fairly well at all the bitrate ranges since end users access the Internet with various bandwidths. Second, error resilience must be integrated into the design of a source coder because error-free transmission under strict delay constraints cannot be ensured. Third, we need to understand the properties of the underlying channel. One interesting observation is that we can actually "modify" the "channel" properties by designing new communication protocols. For example, choosing the Transmission Control Protocol (TCP) instead of the User Datagram Protocol (UDP) to stream video exhibits different "channel" properties. From a source coder point-of-view, a TCP-based transmission approach resembles an underlying error-free channel with time-varying delay and bandwidth [126], while a UDP-based transmission approach resembles a channel experiencing time-varying bit-error rates under fairly constant

delay [93]. Last, we need to design an interface between the underlying network and the video source coder (streaming server); that is, we need feedback from some system function blocks that can estimate and monitor the channel states, and accordingly control the output of the video source coder (streaming server).

1.1 Motivation

The major motivation of this research work comes from the fact that although the application of video streaming has created a new business known as Internet broadcasting, the corresponding research on both video coding schemes dedicated for the purpose of Internet streaming and transport schemes for smooth streaming is still in its infancy [66].

Due to the heterogeneity and dramatic evolvement of the Internet architecture and the transmission-error susceptibility of a compressed video, many problems remain to be solved despite the fact that researchers in video coding and computer network communication areas have separately addressed some problems for many years.

In the author's opinion, there are several areas that have not undergone thorough investigation in current studies on Internet video streaming, some of these areas are highlighted below:

1. The computer networking research community has oversimplified the video source coder model. The work of the network research community on supporting video streaming has been focused on the design of new protocols. The real-time transport protocol (RTP) is the first protocol suite that provides end-to-end delivery services for data with real-time delivery constraints, such as streaming audio and video [106]. The streamed data must be presented to users in appropriate forms in fixed order and at exact scheduled time-spots.

Hence, such transport protocols must provide certain mechanisms to ensure correct presentation. For example, the RTP provides a *sequence number* field and a *timestamp* field in the RTP packet header to ensure correct ordering and scheduling. In recent years, the property of so-called *TCP-friendliness* has become the design objective of many transport protocols that aim at real-time applications over the Internet. A key factor in achieving TCP-friendliness is that there must be a video (we are only concerned with video here) source coder that interacts with the transport protocols and supports their actions. When it reaches the stage that a newly-designed protocol needs performance evaluation, the output of a video source coder is often modelled as packets with layers only distinguished by their bitrates and priority labels [4, 98]. However, modelling the priority hierarchy of video bitstreams generally is not that simple. Fig. 1.3 illustrates a typical priority hierarchy of a scalable video coder. Consequently, studies on fragmentation of compressed videos has been started by the network research community [16].

2. The video coding research community has oversimplified the channel model or has simply employed impractical models. Video coding researchers have traditionally focused on the design of source coding algorithms. In recent years, incorporating error resilience into video coders has become an increasingly popular research topic. However, when it comes to evaluating the performance of an error-resilient video coder, many researchers adopt a simple Markovian model of the packet loss behavior of the underlying channels with only two states, *channel good* and *channel bad*. Nevertheless, a wide body of empirical data argues strongly that the Markovian model may not be suitable for modeling packet losses [91]. Many research topics on robust adaptive video communication are not feasible in the scenario of Internet streaming. For example, feedback-based encoder rate adaption and transcoding normally cannot

be implemented in a streaming server because the server simply cannot deal with a large number of unicast connections demanding such kind of processing. Many error-resilient techniques are designed for bit-oriented transmission, such as resynchronization marker, data partition, and reversible variable-length coding. Depending on these techniques, a decoder can regain synchronization in case of bit errors. But it is obvious that any such technique would become useless if a whole video frame is lost. Unfortunately, this is often the case in very-low-bitrate video streaming when a large maximum transmission unit is used. Hence, the design of packetization (fragmentation) schemes is a key factor that determines the robustness of a streaming system.

3. Fine granular scalability (FGS) is a newly proposed concept for Internet video streaming. The FGS was recently adopted into a profile of the MPEG-4 standard [42] to facilitate the ever-increasing demand for Internet video streaming. Although we have seen the release of the MPEG-4 FGS reference source codes [77], this does not necessarily mean that this should be the concluding stage of research and development on FGS schemes. First, no information on alternative coding frameworks has been made publicly available so far. Second, to the best knowledge of the author, there have been very few publications on the performance evaluation of FGS streaming systems [23].

Based on the justifications above, we believe that further research on error-resilient FGS video source coder and packetization is necessary and an appropriate simulation platform is indispensable in the design and evaluation of the video source coder.

One of the two major problems that we deal with in this thesis is the design of an FGS video coder that has the capability of network-awareness. The concept of FGS was proposed by the Motion Pictures Experts Group (MPEG) committee based on the anticipation that the demand for Internet broadcasting will be enormous in

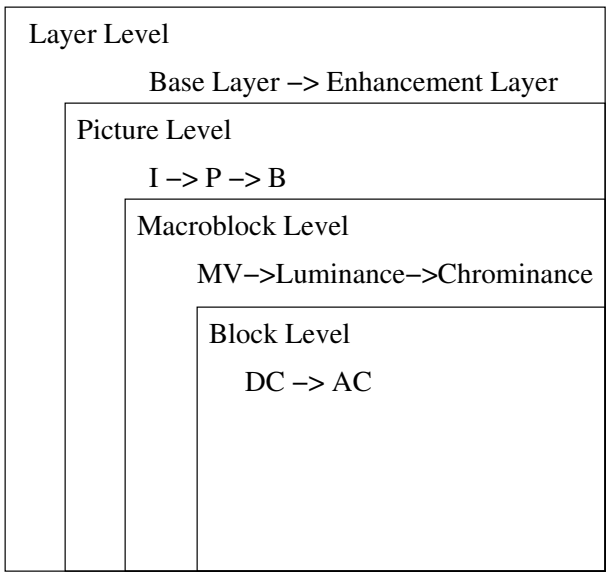


Figure 1.3: The priority hierarchy inside a typical output bitstream of a scalable video coder.

the coming years [65]. Although FGS has been defined in the framework of MPEG-4, which is primarily based on the discrete cosine transform (DCT) hybrid coding framework, the work in this thesis has adopted a discrete wavelet transform (DWT) FGS coding framework based on the following considerations:

- DWT-based still image coding schemes have been shown to be superior to those of the DCT-based JPEG standard [92]. Since the intra-coded frames in MPEG-like codecs are encoded using algorithms similar to those of JPEG, there is little doubt that a DWT-based scheme should offer better performance on the encoding of intra-coded frames or blocks. This judgement is supported by the MPEG-4's adoption of DWT-based sprite coding [43] in video sequences.

- Unlike DCT-based schemes, DWT-based embedded coding schemes can inherently achieve signal-to-noise ratio (SNR) scalability and spatial scalability.

- Most researchers suspect that the DWT-based inter-frame coding schemes cannot compete with the DCT-based counterparts for good reason. But since the

enhancement layer (EL) of an FGS video is encoded as still images, the preference enjoyed by the DCT-based schemes is weakened and is worth further investigation.

The other problem we will consider is the network-awareness of the FGS video encoder. To qualify as having network-awareness, a video encoder must gain knowledge of underlying transport protocols, integrate error-resilience into its output, and packetize the output into datagrams in such a way that they are suitable to be decoded independently.

There has been little work published in the area of wavelet-based FGS video coding, possibly because the FGS is a fairly new concept in a new application scenario and because researchers have gradually lost interest in wavelet video coding based on their experiences with non-scalable coding. Another impending problem is that currently there is no open platform that can provide real-time rate control information to either a source coder or a streaming server, which is essential to both network stability and streaming quality.

1.2 Problem Statement

In this thesis work we investigate: (1) wavelet-based FGS video coding schemes that possess network-awareness, and (2) streaming systems that provide a simulation platform for FGS video streaming.

1.2.1 Network-Aware Wavelet FGS Video Coder

The design of an error-resilient wavelet FGS video coder is the focus of this thesis work. The following issues have been investigated:

- *Motion estimation/motion compensation* (ME/MC): ME/MC techniques contribute significantly to the high compression ratio on raw video sequences. Our

studies have shown that the ME/MC in the wavelet domain is not very efficient. Instead, a spatial domain ME/MC, *i.e.*, ME/MC is conducted before applying 2D-DWT, is adopted in the FGS video coder.

- *Still image/residual frame coding*: Wavelet-based still image coding schemes have undergone in-depth investigation to decrease their computational complexity as well as to maintain good compression performance. The suitability of embedded coding for video coders will be investigated. Because of the different transmission priority of the base layer (BL) and the enhancement layer (EL) of the FGS video, different measures must be taken in the coding of the BL and the EL.

- *Network-awareness*: The source coder must exploit knowledge of the underlying communication protocols. The output of the source coder should be properly fragmented into different transmission-priority groups. The JSCC approaches need to be employed to deal with packet losses.

1.2.2 Streaming System

The design of a streaming system is an integral part of this thesis work so that the performance of the proposed FGS video coder can be evaluated. Due to the limited available resources, the streaming system is designed as add-on modules for the NS-2 network simulator [95]. We constrain the scope of this research project to scenarios where feedback channels in a network are *not* available. Albeit strict and narrow, this assumption is reasonable in several major application scenarios. For example, a busy server may be overwhelmed by a large number of concurrent connections so it is too busy to handle feedback information from all clients. Or, a server in multicast session may end up transmitting lost packets many times for different clients and suffer scalability problems [102]. This assumption also simplifies the design of a

streaming system and makes the performance evaluation of a FGS video coder easier to manage.

The streaming system is designed to evaluate the streaming performance of the proposed FGS video coder and to test the proposed streaming strategy, *i.e.*, protecting the BL of an FGS video with forward error correction (FEC) and protecting the EL with error-resilience incorporated into the source encoder. A mathematical model is established to analyze the optimal employment of FEC on video data.

1.3 Thesis Overview

The main issue addressed in this thesis is the development of an error-resilient wavelet FGS video coder that targets Internet video streaming applications.

The remainder of this thesis is organized as follows. In Chapter 2, we survey related work in the fields of scalable wavelet image and video coding, error-resilient video coding, TCP-friendliness transmission, and Internet video streaming.

In Chapter 3, scalable wavelet image coding algorithms are developed. The focus has been placed on employing simple and efficient context modeling approaches not only to decrease the computational complexity of the coding algorithms but also to maintain a good compression performance. It is evident from our experimental results that well-established simple techniques, if properly deployed, can achieve better cost-performance ratio than complex techniques.

The conceptual simplicity of the algorithms proposed in Chapter 3 directly stimulated our interest to exploit their potential. After gaining insight in the algorithms, we found that the popular progressive bit-plane coding process in embedded image coding can easily be parallelized. This approach is complementary to the conventional local processing and task separation approaches in parallel coding of images. It provides an additional dimension of processing flexibility when embedded wavelet

image coding schemes are parallelized. This parallelization approach is discussed in Chapter 4.

Chapter 5 presents our work in scalable wavelet video coding. Based on the conclusions drawn from our analysis that classical embedded image coding methods are not suitable for motion-compensated residual images, a spatial-scalable wavelet video coder that utilizes the backward ME/MC technique and single-pass estimation-quantization (EQ) coder is developed in §5.1. §5.2 develops the error-resilient wavelet FGS coder whose EL is designed to be network-aware.

Chapter 6 starts by proposing an analytical model for evaluating the application of forward error correction (FEC) in video streaming. The model can be used to determine optimal FEC coding rates in response to different network loss probabilities. Besides the model-based analysis, a basic streaming system is implemented as a set of add-on modules for the NS-2 network simulator. The system uses trace data generated by the proposed wavelet FGS video coder to investigate its streaming performance. Simulation results not only support the validity of the proposed analytical model but also demonstrate the advantages of our proposed streaming strategy.

Finally, in Chapter 7, we summarize the major contributions of this thesis, identify a number of remaining challenges as the future work, and list publications that have directly resulted from the research.

Chapter 2

Related Work

A review of related work is presented in this chapter. §2.1 briefly reviews the discrete wavelet transform (DWT) and its application in still image and video compression. §2.2 introduces traditional scalable video coding techniques as well as the newly proposed FGS coding technique. Several error-resilient techniques used by video encoders are introduced in §2.3. In §2.4, the underlying communication protocols used in video streaming and methods of modeling the bandwidth available to streaming applications are reviewed. §2.5 reviews Internet video streaming schemes and the employment of forward error correction (FEC) technique.

2.1 Wavelet Image and Video Compression

2.1.1 Dyadic Wavelet Transform

The dyadic wavelet transform is an octave-band representation for signals. The discrete wavelet transform (DWT) may be obtained by iterating a two-channel filter-bank on its low-pass output. This multi-resolution decomposition of a signal into its coarse approximation and detail components is useful for data compression, feature extraction, and denoising. In the case of images, the wavelet representation of an

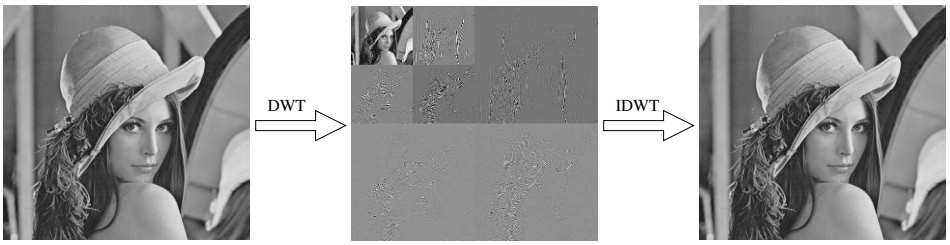


Figure 2.1: Multi-resolution representation of the *Lena* image using DWT.

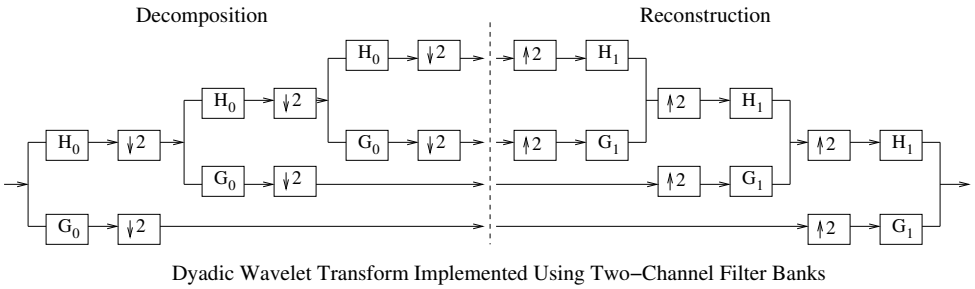


Figure 2.2: One-dimensional dyadic wavelet transform implemented by a two-channel perfect reconstruction filter-bank.

image is well-matched to psycho-visual models, and compression systems based on the DWT yield perceptual quality superior to other methods at medium to high compression ratios [122].

Fig. 2.1 shows an original image, its wavelet representation, and the reconstructed image without coefficient quantization. Since the wavelet transform is invertible, the reconstructed image is identical to the original image. The decomposed image (wavelet representation) shows a coarse approximation sub-image (subband) in the upper left corner and several detail sub-images (subbands) at various scales. As the scale changes, the size of a sub-image changes. This is called "multi-resolution" and is enabled by the decimation operation in the filter-bank structure shown in Fig. 2.2. The coarse approximation and detail images are obtained by filtering the original image by a low-pass filter $H_0(z)$ and a high-pass filter $G_0(z)$, respectively. The filtered images are then decimated by a factor of 2 to preserve the original image size. This is reflected in the structure as a two-channel filter bank.

A *wavelet* decomposition arises from iteration of the low-pass filtering and decimation steps of a multirate filter bank [127]. For a dyadic wavelet decomposition, we iterate on the low-pass output [68, 113]. A finite number of iterations will lead to a discrete-time multi-resolution analysis with low-pass frequency response $\prod_{k=1}^{n} H_0(\frac{\omega}{2^k})$. If the low-pass filter h_0 satisfies the orthonormality constraint, $\sum_k h_0[k] = \frac{1}{\sqrt{2}}$, and has one vanishing moment $\sum_k k h_0[k] = 0$, then the infinite product $\lim_{n\to\infty} \prod_{k=1}^{n} H_0(\frac{\omega}{2^k})$ converges to a function $\phi(\omega)$, whose inverse Fourier transform is the continuous time function $\phi(t)$ called the *scaling function* [68, 113]. The scaling function $\phi(t)$ is the solution to the dilation equation

$$\phi(t) = 2 \sum_k h_0[k] \phi(2t - k) \tag{2.1}$$

and it is orthogonal to its integer translates. If the filter $h_0[n]$ has finite impulse response (FIR), then $\phi(t)$ is said to have compact support. The scaling function determines the *wavelet function* $w(t)$ by means of the high-pass filter $g_0[k]$:

$$w(t) = 2 \sum_k g_0[k] \phi(2t - k) \tag{2.2}$$

The set of dilates and translates $\{w(2^j t - k)\}_{j,k \in Z}$ forms a tight frame (and in most cases an orthonormal basis) for $L^2(\mathbf{R})$ [68].

The set of coefficients $h_0[k]$ is called the set of scaling function coefficients (or the scaling filter). The set of coefficients $g_0[k]$ is called the set of wavelet function coefficients (or the wavelet filter). These two sets of coefficients can be related to each other by imposing conditions. For example, by forcing the orthogonality condition on the integer translates of the wavelets, it is shown in [10] that

$$g_0[k] = \pm(-1)^k h_0[1 - k] \tag{2.3}$$

Alternatively, by requiring a mirror symmetry between the two filters' frequency response, in half-band filter banks [113], the relation between these two filters becomes

$$g_0[k] = \pm(-1)^k h_0[k] \tag{2.4}$$

It is shown in [10] that any continuous function can be represented by the following expansion, defined in terms of a given scaling function and its wavelet derivatives:

$$x(t) = \sum_{k=-\infty}^{\infty} c_{j_0}[k]\phi_{j_0,k}(t) + \sum_{j=j_0}^{\infty}\sum_{k=-\infty}^{\infty} d_j[k]w_{j,k}(t) \tag{2.5}$$

The set of coefficients, $c_{j_0}[k]$ and $d_j[k]$, in the wavelet expansion represented by (2.5) is called the *discrete wavelet transform* (DWT) of the function $x(t)$.

In practice, a discrete signal $x[k]$, at its original resolution is assumed to specify the corresponding scaling coefficients. It is shown in [10] that by assuming $c_J[k] = x[k]$, the DWT coefficients are obtained using the following set of equations:

$$\begin{cases} c_{j-1}[k] = \sum_m h_0[m-2k]c_j[m] \\ d_{j-1}[k] = \sum_m g_0[m-2k]c_j[m] \end{cases} ;\ j = J, J-1, \ldots, j_0+1 \tag{2.6}$$

This calculation is continued until $c_{j_0}[k]$ and $d_{j_0}[k]$ are determined. Then the set of these coefficients, namely, $\{d_{J-1}[k],\ d_{J-2}[k],\ \ldots,\ d_{j_0+1}[k],\ d_{j_0}[k],\ c_{j_0}[k]\}$, is called the DWT of the original signal $x[k]$.

It is also shown in [10] that the higher resolution scaling coefficients are related to the lower resolution scaling and wavelet coefficients by the following relationship.

$$c_{j+1}[k] = \sum_m c_j[m]h_0[k-2m] + \sum_m d_j[m]g_0[k-2m] \tag{2.7}$$

This equation shows how the DWT sequences at resolution j_0 can be used iteratively to reconstruct the scaling coefficients at the highest resolution, J.

There are two major classes of wavelet transform systems. One class consists of orthogonal wavelets and the other one consists of bi-orthogonal wavelets. In the orthogonal wavelet systems, knowledge of the scaling function is sufficient for the design of analysis and synthesis filters [113]. For a given even-sized FIR scaling factor, $h[k]$, we have

$$\begin{cases} Lowpass\,analysis: & h_0[k] = h[k] \\ Lowpass\,synthesis: & h_1[k] = h_0[K-1-k] \\ Highpass\,analysis: & g_0[k] = (-1)^k h_1[k] \\ Highpass\,synthesis: & g_1[k] = -(-1)^k h_0[k] \end{cases} \tag{2.8}$$

The orthogonality requirement results in a complicated design and precludes the use of linear phase filter banks that are desired in image and video processing. By allowing dual-basis in bi-orthogonal wavelets, it is easier to achieve greater flexibility.

One pair of scaling functions and corresponding filters are introduced in [18, 113]. Assume that $h[k]$ and $\tilde{h}[k]$ represent the pair of scaling filters. They should satisfy

$$\sum_k \tilde{h}[k]h(k-2l) = \delta(l) \tag{2.9}$$

where $\delta(l)$ is the Dirac delta function. There are many different approaches to the design of bi-orthogonal wavelets. In almost all of them, the requirement is to make the resulting FIR filters to have a linear phase. To attain the perfect reconstruction property [113], the following conditions should be satisfied.

$$\begin{cases} Lowpass\,analysis: & h_0[k] = h[k] \\ Lowpass\,synthesis: & h_1[k] = \tilde{h}[k] \\ Highpass\,analysis: & g_0[k] = (-1)^k h_1[k] \\ Highpass\,synthesis: & g_1[k] = -(-1)^k h_0[k] \end{cases} \tag{2.10}$$

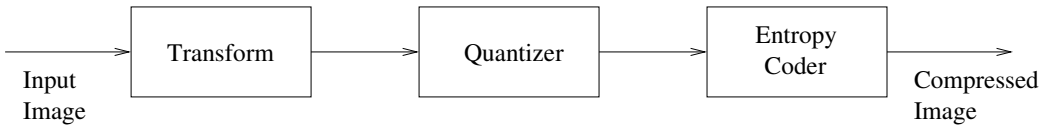


Figure 2.3: A typical transform-based lossy image encoder.

2.1.2 Wavelet Image Coding

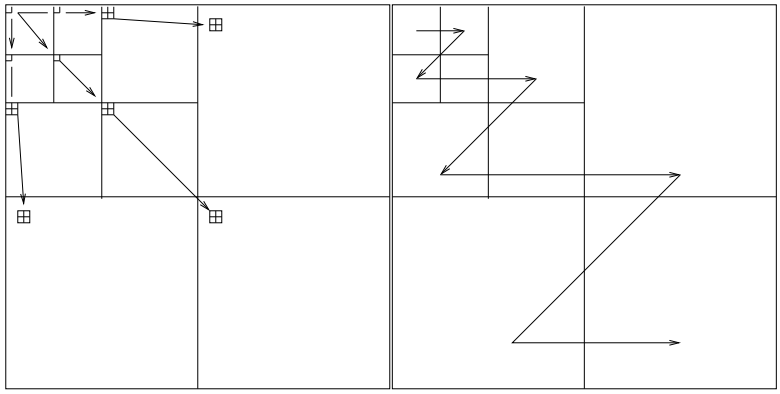
A typical lossy image compression system is shown in Fig. 2.3. It consists of three closely connected components. Compression is accomplished by applying a linear transform to decorrelate the image data, quantizing the resulting transform coefficients, and entropy-encoding the quantized values.

Since the appearance of DWT, many enhancements have been made to the standard quantizers and encoders to exploit the DWT's multi-resolution capability, the properties of the human visual system, and the statistics of the transform coefficients. These have led to improved results in terms of higher compression ratios.

Over last decade, a variety of novel and sophisticated wavelet-based image coding schemes have been developed. Among those representative schemes are EZW [108], SPIHT [105], SFQ [140], ECECOW [139], WDR [121], EBCOT [118], etc. This list is by-no-means exhaustive and many more innovative techniques are being developed as this thesis was being written.

In wavelet decomposition, shown in Fig. 2.4(a), each coefficient in the high-pass subbands, except for the three subbands at the finest scale, has four coefficients corresponding to its spatial position in the next finer scale. Because of this very structure of the DWT decomposition, Lewis and Knowles [58] first introduced a tree-like data structure to group the wavelet coefficients.

Later, Shapiro [108] called this structure a *zerotree* of wavelet coefficients, and presented his embedded zerotree wavelet (EZW) algorithm. The zerotree is based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with



(a) (b)

Figure 2.4: (a) Zerotree data structure and (b) subband scanning order.

respect to a given magnitude threshold T, then all wavelet coefficients of the same orientation in the same spatial location at a finer scale are also likely to be insignificant with respect to T. The EZW algorithm encodes the tree structure and orders the bits that are generated in order of importance, yielding a fully embedded bitstream. The main advantage of EZW coding is that the encoder can terminate the encoding at any point, thereby allowing a target bitrate to be met precisely. Similarly, the decoder can also stop decoding at any point and reconstruct an image. The algorithm produces excellent results without any pre-stored tables or codebooks, training, or prior knowledge of the image source.

Since the inception of the EZW algorithm, many techniques have been proposed to improve the efficiency of the EZW. An extremely popular variation of the EZW is the set partitioning in hierarchical trees (SPIHT) algorithm proposed by Said and Pearlman [105]. They proposed a new and more effective implementation of the modified EZW algorithm based on set partitioning in hierarchical trees, and called it the SPIHT algorithm. They also presented a scheme for progressive transmission of the coefficient values that incorporates the concepts of ordering the coefficients by magnitude and transmitting the most significant bits first. The SPIHT algorithm

performs well even without the entropy coding stage.

The new-generation still image coding standard, JPEG-2000 [40], is based on embedded block coding with optimized truncation of the embedded bitstreams (EBCOT) [118]. The EBCOT algorithm is closely related to earlier work on scalable image compression, such as EZW and SPIHT.

Rather than focusing on generating a single scalable bitstream to represent an entire image, the EBCOT partitions each subband into relatively small blocks of coefficients and generates a separate scalable bitstream to represent each so-called "code-block". The algorithm uses the context modeling approach extensively to encode magnitude and sign information of the coefficients. The entropy coding of the EBCOT resembles the arithmetic coder proposed in [30]. The algorithm exhibits the state-of-the-art compression performance while producing a bitstream with an rich feature set, including resolution and SNR scalability together with the random access property. The algorithm has modest complexity and is extremely well suited to applications involving remote browsing of large compressed images [118].

A drawback of the zerotree-based algorithms is that they only implicitly encode the positions of significant coefficients. This makes it difficult to perform operations which depend on the exact position of significant coefficients, such as region selection on compressed data. Such operations are possible with the wavelet difference reduction (WDR) algorithm of Tian and Wells [121]. The term "difference reduction" refers to the way in which the WDR encodes the locations of significant wavelet coefficients. Although the WDR does not typically produce higher PSNR results than the SPIHT, its inventors claim that the WDR can produce perceptually superior images at very low bitrates. Except that WDR explicitly encodes the positions of significant coefficients, WDR adopts the principle of bit-plane coding and set partitioning proposed in EZW and SPIHT.

In contrast to all the above schemes which do not explicitly utilize the statistics of

the decomposed wavelet subbands, many wavelet image coders model the coefficients in wavelet subbands as random variables drawn from a mixture of distributions. This is based on the observation that subband coefficients have highly non-Gaussian statistics [68].

Universal context modeling approaches have been investigated in [14, 64, 139]. These works model the statistics of the wavelet subband coefficients as non-stationary generalized Gaussian, whose non-stationarity is manifested by a slowly varying variance (energy) in each subband. Because the energy varies slowly, it can be predicted from causal neighboring coefficients. Therefore, these algorithms attempt to recover the variance at the decoder from already transmitted data. Based on the estimated variance, an optimal scalar quantizer can be chosen from a look-up table that is pre-computed and available at both encoder and decoder. Finally, the quantizer bin probability corresponding to the quantized symbol will be used by an arithmetic coder to encode each symbol.

2.1.3 Wavelet Video Coding

The application of the DWT in still image coding has proved to be popular and successful as it was adopted in the latest still image compression standard, JPEG-2000 [40]. It is natural that researchers wish to broaden its scope to the area of video coding. There are several major reasons that the DWT is desirable in video coding.

- The computational complexity of motion estimation (ME) can be reduced since hierarchical ME can be applied.
- Wavelet coding algorithms can exert bit-wise control on the bitrate of a scalable compressed bitstream. Hence, SNR scalability can be more easily achieved.
- Spatial scalability can be achieved inherently because of the multi-resolution feature of the decomposed wavelet pyramid.

- The blocky artifacts experienced in DCT-based video coding schemes do not occur in wavelet-based coders.

A trivial extension of wavelet image coding to video is to encode each video frame independently as in Motion JPEG-2000 [39], where the frames in a sequence are independently encoded using the JPEG-2000 [40] image compression standard. But this results in a poor compression ratio because such a scheme does not exploit the tremendous temporal redundancy in typical video sequences.

There are three fundamental coding architectures employed in wavelet video coding schemes:

1. *In-band prediction*: The DWT is first performed on each original frame. Temporal redundancy is then exploited in subbands by ME/MC techniques.

2. *Wavelet in loop*: A conventional hybrid coding architecture is used as in the MPEG series video coding standards. The DCT is replaced by the DWT to encode the residual frames in the motion compensation (MC) prediction loop.

3. *Spatio-temporal filtering*: Wavelet filtering along the temporal axis followed by 2-D spatial DWT, or vice versa. ME/MC is optional depending on motion activity in the video sequences. This is also called 3-D wavelet video coding.

The first coding architecture tries to exploit the multi-resolution characteristic of the wavelet pyramid. A video frame is first decomposed into a pyramid consisting of subbands at different scales. Motion vectors are first estimated at the coarsest scale. The resultant motion vectors are then used as approximations for motion vectors at finer scales. The approximated motion vectors are used to narrow the search region for motion vectors at finer scales. Thus, the computational complexity of ME/MC is reduced. The motion vectors for each scale are hierarchically computed from coarser scale to finer scale. The residual subbands are quantized and entropy coded [70, 141, 143].

Techniques in the second category follow the traditional hybrid coding architecture. Inter-frame redundancy is first removed by employing an ME/MC stage. The residual frames are then encoded by wavelet image coding techniques that address the characteristics of motion-compensated residual images. These techniques provide smoother transition from the DCT-based schemes to the DWT-based schemes. They normally focus more on the quantization and entropy coding of residual images rather than on ME/MC techniques [72, 110, 131]. Since these techniques adopt the same architecture as the traditional DCT-based coding schemes, it is possible to incorporate the DWT-based techniques into current DCT-based video coders [60].

The third architecture employs temporal filtering as well as spatial decomposition. It is also called 3-D wavelet video coding. Karlsson and Vetterli were the first to extend subband coding techniques to video by introducing a spatio-temporal wavelet pyramid [50]. The simplest example of a temporal wavelet filter is the Haar wavelet. In this case, the Haar decomposition creates two new frames that represent the average and the difference of two neighboring original frames. If there is little motion activity in the original frames, the difference frame contains mostly zero values and compresses well. If there is high motion present, then ME/MC are performed to construct motion trajectories for each pixel in the original frames. The temporal filtering is conducted on the motion trajectories. The temporally filtered frames are then spatially decomposed into wavelet pyramids. These pyramids can be encoded separately or be encoded together using 3-D coding techniques [54, 119, 136].

2.2 Scalability in Video Coding

The objective of traditional video coding schemes has been to optimize the video quality *at a given bitrate*. In these schemes, the encoders compress the input raw video signal into a bitstream with a rate that is lower than the channel capacity.

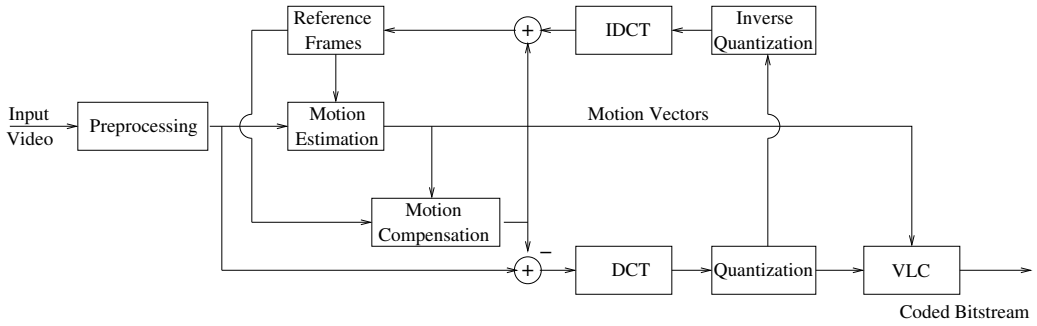


Figure 2.5: Generic DCT-based hybrid video encoder [6].

This is based on two assumptions. The first is that the channel capacity is known. The second is that the decoder can receive the bitstream in time to decode the video. A block diagram for generic video encoder is shown in Fig. 2.5.

When compressed video is transmitted over channels that experience intermittent error bursts, researchers encounter the problem of how to maximize the utilization of the available bandwidth. If the full bandwidth is used to transmit the compressed video, severe visual quality degradation is unavoidable since errors propagate in the decoded video frames because of the decoding dependency in the compressed video. If enough redundancy is provided by channel coding, effective bandwidth utilization lowers. Researchers came up with a trade-off solution: only a small portion of output bitstream (usually called *base layer*), which can be decoded to reconstruct video with mediocre quality, is protected by channel coding. The remaining portion (usually called *enhancement layer*) is used to enhance visual quality, is left unprotected without channel coding. In this way, we ensure a better-than-mediocre visual quality most of the time without sacrificing much bandwidth. This is the main principle of *scalable coding* techniques [3, 31, 115].

When we talk about scalable video coding, usually three categories of scalability are indicated: signal-to-noise ratio (SNR) scalability, temporal scalability, and spatial scalability. Recently, a new type of video scalability, namely, fine granular scalability (FGS), has been proposed following the rapid growth of Internet video streaming.

These scalable techniques are briefly reviewed in this section.

2.2.1 SNR Scalability

SNR scalability is a technique to encode a video sequence into layers (normally two) at the same frame rate and the same spatial resolution, but with different quantization accuracy. Fig. 2.6 shows the SNR scalability decoder defined in the MPEG-2 video coding standard [41]. The base layer (BL) bitstream is decoded by the BL variable-length decoder (VLD) first. The dequantization in the BL produces the reconstructed DCT coefficients. The enhancement bitstream is decoded by the VLD in the enhancement layer (EL) and the enhancement residues of the DCT coefficients are produced by dequantization in the EL. A higher accuracy DCT coefficient is obtained by adding the BL reconstructed DCT coefficient and the EL DCT residue. The DCT coefficients with a higher accuracy are given to the inverse DCT (IDCT) unit to produce reconstructed image domain residues that are to be added to the motion-compensated block from the reference frame. One problem with this architecture is that the EL is involved in the prediction loop. Whenever the EL bitstream is unavailable, a reconstructed frame will be different from what it is meant to be and the difference is called *drift error*. One solution to reducing drift error is to exclude the EL from the prediction loop, which will increase the bitrate of the EL in the mean time. The other solution is to construct two prediction loops in both the BL and the EL to reduce the bitrate of the EL [32].

2.2.2 Temporal Scalability

Temporal scalability is a technique to encode a video sequence into two layers at the same spatial resolution, but at different frame rates. The BL is coded at a lower frame rate. The EL provides the missing frames in BL to form a video sequence with a higher frame rate. Coding efficiency of temporal scalability is high and very close

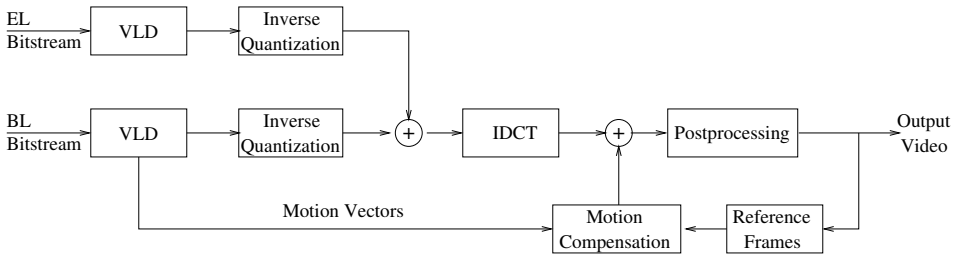


Figure 2.6: SNR scalability decoder architecture defined in MPEG-2.

to non-scalable coding. The reason is that temporal scalability can be achieved easily without modifying generic coding architecture defined in video standards, e.g., B-frames (which are not used as reference frames and could be discarded) in the coded bitstream can be seen as one form of EL in temporal scalability. Since B-frames are not used in prediction loop, errors in B-frames do not influence the decoding of any other frame. P-frames can also be used as EL as long as they are not in the prediction loop, *i.e.*, employed to predict future frames.

2.2.3 Spatial Scalability

Spatial scalability is a technique to encode a video sequence into two layers at the same frame rate, but at different spatial resolutions. The BL is coded at a lower spatial resolution. The reconstructed BL frame is up-sampled to form the prediction for the high-resolution frame in the EL. The difference between the original frame and the predicted frame is then encoded into the EL bitstream [79]. Instead of filtering and scaling video frames in the spatial domain, similar operations can also be conducted in the compressed domain and realize spatial scalability [38, 76]. If the spatial resolution of the BL is the same as that of the EL, *i.e.*, if the up-sampling factor is one, the spatial scalability decoder can also be considered as an SNR scalability decoder. Since the DWT can decompose a video frame into a set of subbands with different spatial resolutions, spatial scalability can be achieved inherently in

wavelet-based video coding schemes [141].

2.2.4 Fine Granularity Scalability

Since the Internet is a heterogeneous network and end users possess devices with different bandwidths and computing powers, neither the channel capacity nor the transmission delay can be determined when a raw video is encoded and stored. Consequently, it is preferable to have an encoder that outputs a single bitstream, which can be decoded by various devices to reconstruct video at more than one layer of resolution and visual quality. A common feature of the layered coding techniques discussed in §2.3.3 is that the bitstream for a specific layer needs to be received completely without error, otherwise, annoying visual artifacts may drastically downgrade visual quality. A major problem with layered coding is that the quality of the reconstructed video is limited by the maximum integer number of layers that can be accommodated by the available bandwidth. For example, suppose we have a BL coded at bitrate R_B, an EL coded at bitrate R_E, and available instantaneous bandwidth at R_i. If $R_i \geq R_B + R_E$, both layers can be decoded. However, the decoder can only decode the BL completely if $R_B < R_i < R_B + R_E$. Whether the extra portion of EL can be recovered with some useful information is not guaranteed. This is not an optimal solution since there is a high probability that the bandwidth of $(R_i - R_B)$ is wasted. To address this problem, a new type of scalability called fine granularity scalability (FGS) is defined in a new profile of the MPEG-4 [42]. With the FGS video coding, we can decode any portion of the enhancement layer bitstream and progressively enhance the visual quality as seen in Fig. 2.7. In the MPEG-4 FGS coding, the error frame between the original input frame and the reconstructed BL frame constitutes the enhancement layer to be DCT-transformed and encoded by the bit-plane variable length coder (VLC) as shown in Fig. 2.8. For each 8×8 error block of DCT coefficients, the absolute values of the 64 coefficients are

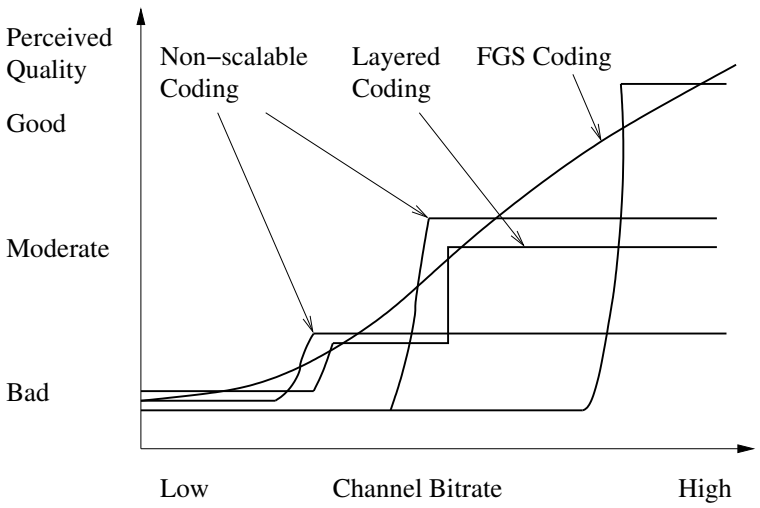


Figure 2.7: Performance of different video coding schemes.

zigzag-ordered into an array. Each bit-plane of the 64 coefficients is variable-length coded [59]. The EL of an FGS video consists of the error data between the BL and the original sequence and is encoded as still images. Some researchers have proposed to include a prediction (ME/MC) loop in the coding of the EL to increase compression ratio [130, 137]. The schemes that involve motion compensation in the EL coding act more like a traditional scalable coding scheme except that fine granular scalability is still preserved. However, adopting a prediction loop in the EL increases the computational complexity and makes the bitstream less robust because of the data dependency incurred by the prediction loop.

2.3 Error Resilience

Traditional data communication applications cannot tolerate errors during transmission since the corrupted text files or executable programs are typically useless for the receiver. Usually, techniques like forward error correction (FEC) and automatic retransmission request (ARQ) can be adopted to correct transmission errors. Because video communication has some unique characteristics, careful analysis is needed be-

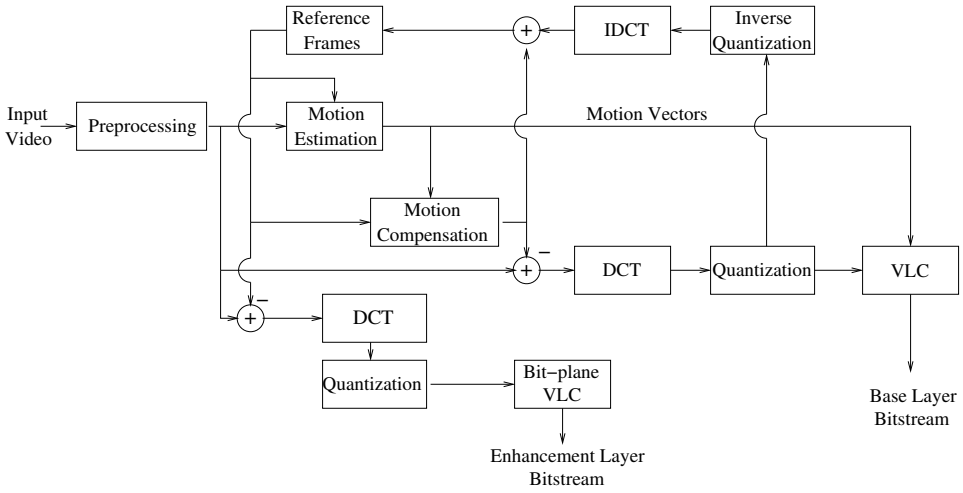


Figure 2.8: FGS encoder architecture [59].

fore we employ traditional error protection techniques. It is common sense that a video with varying delay-jitter is annoying, especially in teleconferencing. Also, a small area of mosaic artifacts on the TV screen lasting for only a few seconds is tolerable considering the area of the full screen and the total length of the video. Thus, video communication systems demand real-time delivery while allowing a certain amount of transmission error. Since error-free transmission is not required by video communication systems, researchers prefer not to use ARQ because it cannot satisfy the real-time constraints in most cases. In some low bandwidth time-varying channels it is even hard to employ FEC, not to mention choosing a channel coding rate. Therefore, the JSCC approach has been advocated in recent years, which allocates a certain level of redundancy at the source coding stage. All *error-resilient* coding techniques work under this premise and intentionally make the source coder less efficient than it can be, so that transmission errors will not cause disastrous effects on the decoded video. The counterpart methods at the receiver side are called *error-concealment* techniques [133], which utilize the decoded data to predict the information lost during transmission. In this thesis, we focus on the error resilient coding techniques at the sender side. We now present a brief overview of frequently

used error resilient techniques.

2.3.1 Robust Entropy Coding

The main cause of the compressed video's susceptibility to transmission errors is the usage of variable length code (VLC) in compression. By using VLC, each bit in the compressed bitstream needs to be correctly received for synchronization between encoder and decoder. One single bit error will cause the loss of synchronization and leave the following bits undecodable.

One simple and popular technique is to insert synchronization markers periodically into a bitstream for resynchronization. Using this technique, we still lose the information between an erroneous bit and the next synchronization marker. To recover this information, another technique, namely, reversible variable length coding (RVLC), is proposed to decode the bitstream in a backward manner as shown in Fig. 2.9. Hence, part of the information deemed as undecodable before can now be recovered [114]. These two techniques are adopted in the H.263 and the MPEG-4 video coding standards [43, 45]. Instead of explicitly inserting synchronization markers, synchronization can also be maintained by fragmenting variable-size code blocks to fit them into fixed-size slots as shown in the error-resilient entropy code (EREC) technique [97]. The EREC is especially useful when the more important information is transmitted near the start of each variable-size code block and is not dependent on following data.

2.3.2 Multiple Description Coding

Multiple description coding (MDC) assumes that there are several parallel channels between a sender and a receiver, and each channel may be temporarily down or suffering from bursty errors (Fig. 2.10). With MDC, several coded bitstreams are generated and transmitted simultaneously over separate channels. At the receiver

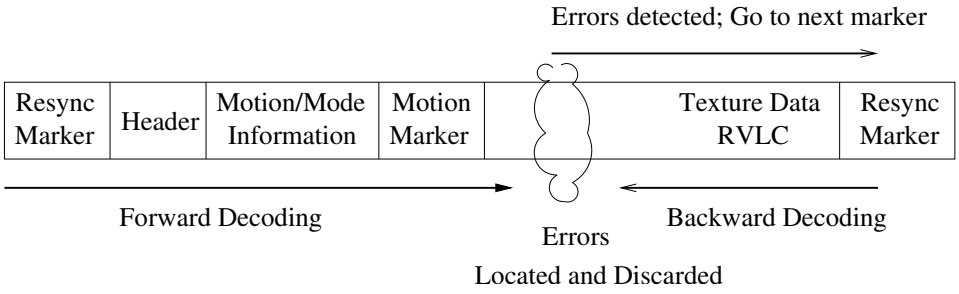


Figure 2.9: The syntax of the RVLC word in the MPEG-4 standard

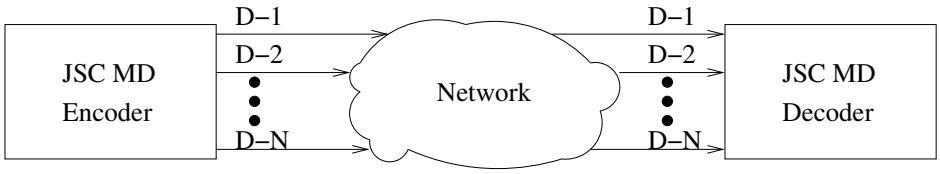


Figure 2.10: Generic block diagram for multiple description coding. D-X represents the X-th description of the coded data.

side, depending on which descriptions are received correctly, different reconstruction schemes are invoked. The MDC encoders and decoders are designed such that the quality of the reconstructed video is acceptable with any description and that incremental improvement is achievable with more descriptions. To guarantee an acceptable quality with a single description, each description must carry sufficient information about the original signal. This implies that there will be overlap in the information contained in different descriptions. Obviously, this will reduce the coding efficiency compared to the conventional single description coder that aims at achieving the highest compression ratio [128].

2.3.3 Layered Coding with Unequal Error Protection

The most popular and effective error-resilient scheme for video transmission has been layered coding combined with transport prioritization. In layered coding, the information is partitioned into multiple layers [31, 96]. A typical method is to partition the information into two layers: a base layer (BL) and an enhancement layer (EL).

The BL is assigned high transport priority so that it is delivered with higher degree of error protection. Layered coding can be implemented in several different fashions depending on how the video information is partitioned. When the partition is performed in the temporal domain, the BL contains a lower frame-rate video while the EL can be added to reconstruct a higher frame-rate video. Layered coding can also be performed in the spatial domain. Wavelet image and video coders are very efficient at generating multi-resolution layers. The progressiveness of most wavelet coders offers the flexibility to generate layers. After the generation of layered bitstreams, FEC is applied to source bitstreams for error protection. Since the BL is the basic component in reconstruction of compressed video, it requires a higher level of error protection. By varying the channel coding rates for different layers, unequal error protection (UEP) is realized [1].

2.4 Network Characteristics

Streaming compressed video data smoothly over packet networks requires knowledge of the characteristics of the underlying networks. Since this thesis is built upon the Internet Protocol (IP) architecture [125], we present an overview of this architecture to provide background for later discussions.

IP architecture defines the mechanisms and protocols for delivering messages in an end-to-end manner in an interconnected "network of networks", or the Internet. The fundamental message delivery unit in IP is called a *datagram*, while the fundamental transmission unit is called a *packet*. Datagrams are usually encapsulated directly in packets, but if the size of a datagram exceeds that of the maximum transmission unit (MTU) [78] of the underlying network, the datagram is *fragmented* into multiple packets. The fragmented packets will be reassembled at the destination host.

An IP *router*, which has two or more network interfaces, is used to *switch* or *for-*

ward packets between networks. All routers within the Internet cooperate to deliver packets from an arbitrary sender to an arbitrary receiver. Each router maintains a table of *routes* that maps a destination network to an outgoing interface. When a packet arrives, the router locates the proper route and forwards the packet toward its destination on the corresponding interface.

The traditional Internet service model is *best-effort*. The network does not guarantee that packets necessarily reach their destination, nor does it guarantee that packets are delivered in the order they were sent. Routers are designed to make a reasonable, best-effort attempt at delivering packets toward the packets' destination. Because packet loss is not precluded by this service model, applications or transport protocols built on top of IP must be made robust to such loss.

Since each incoming packet is temporarily stored in the buffer of a router before the router locates an outgoing interface, the finite size of the router buffer forces the router to drop packets when congestion occurs. Thus, measures must be taken somewhere above the IP protocols (Network Layer) to handle packet loss under heavy network traffic. The basic algorithm incorporated by Transmission Control Protocol (TCP) [126] to handle packet loss is *Congestion Avoidance* [46]. The congestion avoidance algorithm probes for available bandwidth by slowly increasing a congestion window (used to control how much data is outstanding in the network). When congestion is detected (indicated by the loss of one or more packets), TCP reduces the congestion window by half. This rapid back-off in response to congestion is the key to TCP's success in avoiding congestive collapse of the Internet. However, the usage of TCP in multimedia streaming is problematic because:

- The additive increase multiplicative decrease (AIMD) based congestion avoidance algorithm used by the TCP results in drastic fluctuation of the sending rate. If the receiver chooses to decode as soon as the datagram arrives, the user may experience the rapid variation in the visual/aural quality of the de-

coded video/audio. Another option is to use a large buffer to smooth out the variation, but then the interactivity is decreased. In both cases, the user can observe the degradation of the service.

- TCP ensures strict reliability and ordering semantics at the cost of end-to-end delays and delay jitters. For streaming applications, end-to-end delays and delay jitters are among the important factors for streaming quality evaluation. Humans can tolerate a certain amount of errors and losses but long playback delay and delay jitter is very annoying.

- TCP, being purely window-based, results in burstiness in data transmission. This burstiness makes it hard to maintain the relative temporal ordering of the various frames of data between a sender and a receiver.

Because of the problems mentioned above, User Datagram Protocol (UDP) [93] is commonly used as transport layer protocol for streaming applications. Since the UDP does not guarantee the delivery of packets, a receiver needs to take some measures to detect packet loss. This is accomplished by an upper transport layer protocol, *Real-time Transmission Protocol* (RTP) [106]. The RTP is designed to provide end-to-end transport functionalities for supporting real-time applications. The RTP standard consists of two elementary services that are transmitted over two different channels. One of them is the Real-time Transport Protocol which carries the data and the other works as a control and monitor channel called the *Real-time Transmission Control Protocol* (RTCP).

The RTP protocol suite itself does not guarantee quality of service (QoS) or reliable delivery, rather, it provides the following mechanisms in support of media streaming:

- *Time-stamping:* The RTP provides time-stamping to synchronize different media streams. However, it is not responsible for the synchronization, which is

handled by applications.

- *Sequence numbering:* Since packets arriving at the receiver may be out of temporal order, the RTP employs sequence numbering to help place the incoming RTP packets in correct order. It is also used to detect packet loss.

- *Payload type identification:* The type of the payload contained in an RTP packet. The receiver interprets the content of the packet based on the payload type identifier.

- *Source identification:* The source of each RTP packet is identified by a synchronization source identifier (SSRC), which provides a means for the receiver to distinguish individual source from multiplexed sources.

RTCP is an integral part of the RTP protocol suite, which is designed to convey control information such as packet loss ratio, cumulative number of lost packets, and packet inter-arrival jitter.

The relationship among the protocols used for media streaming is illustrated in Fig. 2.11. The Real-time Streaming Protocol (RTSP) [107] and the Session Description Protocol (SDP) [36] belong to the category of *session control protocols* that is responsible for the establishment of sessions and the controlled delivery of media data.

Since general transport protocols (UDP and RTP) used by streaming applications do not offer any congestion control mechanism, congestion control must be integrated into streaming applications. In addition, since a large portion of traffic on the Internet is TCP-based, the designed congestion control algorithm must ensure that TCP connections using AIMD get their fair share of bandwidth in the presence of streaming applications and vice versa. Such algorithms are called *TCP-friendly*. In practice, streaming applications first need to collect and monitor the network characteristics such as MTU, round-trip time (RTT), and packet loss rate. These

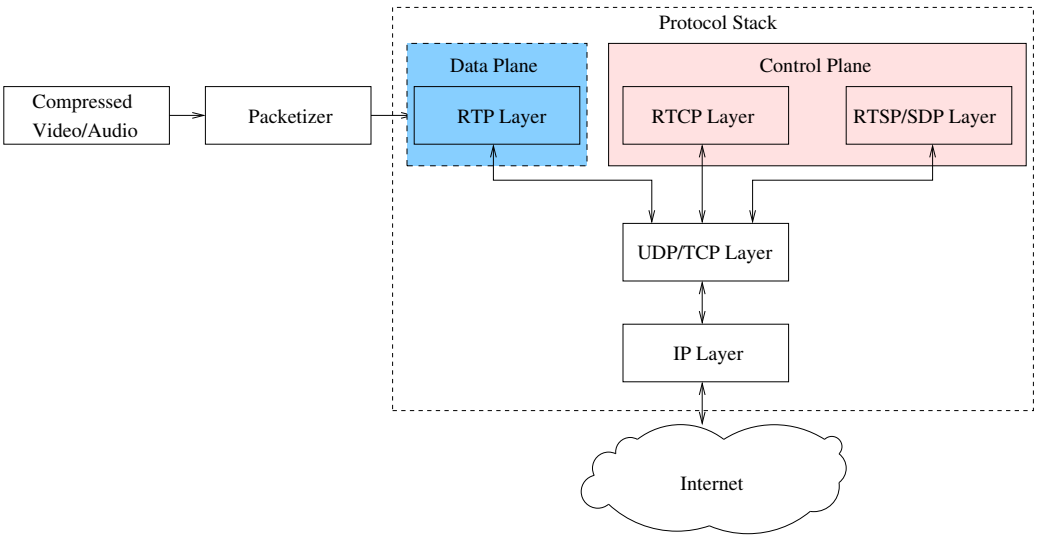


Figure 2.11: Protocol stacks for media streaming.

parameters are then used to estimate the current and future bandwidth. The estimated bandwidth will be used as an input to the congestion/rate control functional block in streaming applications [67, 88].

2.5 Internet Video Streaming

As we have mentioned before, the Internet is a heterogeneous network of networks. Currently, or in the foreseeable future, ensured QoS transmission [111] of video over the Internet is not feasible. Transmission error in Internet video streaming is embodied as lost packets. The cause of packet losses can be attributed to either loss during transmission or severe delay because packets received after the scheduled decoding time are useless. Studies on video streaming have shown that packet losses are inevitable during transmission over public networks. Because of the temporal and the spatial dependency in compressed video data, the effects of lost packets tend to persist for many frames [8]. Hence, streaming compressed video over the public Internet without any type of error protection is not recommended. A well-established principle for error-resilient video streaming over the Internet is application layer framing

(ALF) [17], in which video data must be fragmented consistently with the syntaxes of the underlying encoding algorithms. This ALF principle has guided the design of RTP payload formats for streaming video data that are compressed by various video coding standards [37, 52, 124, 144].

An important parameter for Internet video streaming is the instantaneous *available bandwidth*, which specifies the upper limit of the transmission rate of a sender at any given time. Although accurate estimation of this limit is difficult because of its time-varying characteristic, several studies have introduced metrics, techniques, and tools to do the work [5, 48, 88, 94]. Techniques that adapt their rates have been reported in the literature [7]. To make the Internet video streaming applications share bandwidth fairly with the predominantly TCP traffic, rate control techniques have been proposed to let the network multimedia applications achieve so-called TCP-friendliness [27, 63, 75, 98, 101]. It should be noted that rate control techniques are still in active research and there is no globally accepted solution.

Delivery of video packets in real-time differs fundamentally from delivery of data packets in that video packets have an obsolete time limit. Whenever there is not enough time for retransmission of lost video packets, proactive measures, such as forward error correction (FEC), tend to play an important role in offering error protection. The FEC technique is also important in multicast applications [82, 100] where different clients may experience different loss patterns and retransmission for each client becomes extremely expensive. Several studies have reported employing the layered FEC in providing error resilience for streaming layered video [13, 57, 116, 129]. In these schemes, different layers of the scalable video are protected by different rates of FEC according to their levels of decoding priority.

Chapter 3

Efficient Wavelet Based Still Image Coding

One of the most successful applications of wavelets is transform-based still image compression. A wavelet-based image coder applies a 2D-DWT to an image to remove redundancy, quantizes the wavelet coefficients, and finally entropy-encodes the quantized symbols.

§3.1 reviews a very simple and efficient embedded wavelet still image coding method, which is proposed and named *wavelet difference reduction* (WDR) by Tian and Wells [121]. In §3.2 we show that the timing performance of the WDR can be enhanced without sacrificing compression performance. We propose a context modeling approach to significantly improve the performance of the WDR in §3.3. The chapter ends in §3.4 with a novel coding scheme that applies context modeling to color images. The proposed scheme performs better than the latest adopted JPEG-2000 standard [39] in terms of color SNR and PSNR.

3.1 Wavelet Difference Reduction (WDR) Image Coding

In this section, we describe the original WDR algorithm proposed by Tian and Wells [121]. We will show that WDR borrows many ideas from the more popular EZW [108] and SPIHT [105], while adopting a totally different data structure that exploits the statistical dependency among the wavelet coefficients.

Both the EZW and the SPIHT algorithms utilize a zerotree data structure and bit-plane coding to efficiently encode the wavelet coefficients. The zerotree data structure was first proposed in [58] to exploit the cross-scale dependency by encoding the quantized-zero symbols jointly. Especially in the low bitrate coding situations, most wavelet coefficients are quantized to zero since the quantization step-size is large. This makes the zerotree a very powerful data structure for compression. In the EZW, Shapiro combines it with bit-plane coding to produce an embedded bitstream. The SPIHT is very similar to the EZW in its use of zerotrees, but the SPIHT uses a different approach in the coding of zerotree information and a different ordering of output bits. The SPIHT can achieve good compression without an entropy coding stage. The SPIHT conducts a *sorting pass* and a *refinement pass* recursively from the highest bit-plane to lower bit-planes. Within each bit-plane P, the wavelet coefficients are partitioned into three lists according to a "significance test". A wavelet coefficient c is labeled as *significant* if $|c| \geq 2^P$. The three lists are entitled list of insignificant sets (LIS), list of insignificant pixels (LIP), and list of significant pixels (LSP).

The similarity of the encoding stages between the WDR and the SPIHT is quite conspicuous. The WDR also conducts a sorting pass and a refinement pass for each bit-plane. As the counterparts of the three lists in the SPIHT, three sets are defined in the WDR, namely, a set of insignificant coefficients (ICS), a set of significant

coefficients (SCS), and a temporary set of significant coefficients (TPS). Since the WDR does not utilize the zerotree data structure, it does not have a set like the LIS in the SPIHT. Instead of directly concatenating the significant coefficients found in a given bit-plane to SCS, like the SPIHT adding such coefficients to the LSP, the WDR adds newly-found significant coefficients to the TPS. Afterwards, the TPS is concatenated to the end of the SCS when the refinement pass is finished.

The only difference between the WDR and the SPIHT is in the sorting pass. Instead of using zerotrees to represent insignificant coefficients, the WDR defines a scan-order of wavelet coefficients that passes through all subbands in a wavelet pyramid from the highest level to lower levels. The scan-order maps a 2-D wavelet pyramid into a 1-D array of wavelet coefficients. A significance test is conducted on the 1-D array to pick out the significant coefficients. By encoding the positions of significant coefficients with run-length coding (RLC), the output from the sorting pass consists of the signs of significant coefficients along with a sequence of bits that encodes the position of significant coefficients.

As an example, suppose we have a 4×4 subband. First we map it into a 1-D array of 16 coefficients. If the 1st, the 8th, and the 14th coefficients are significant compared to a given threshold T, we need to output the signs of these significant coefficients and the positions of the significant coefficients. In this example, the number of insignificant coefficients between each pair of neighboring significant coefficients are 1, 7, and 6. Since the most significant bit (MSB) of these numbers is always 1, there is no need to output the MSB. So we output nothing for 1, output "**11**" for 7, and output "**10**" for 6. In order to indicate the end of an output number, the signs of the significant coefficients are inserted to mark the end of output for each significant coefficient. If the signs of the three significant coefficients are "$+\ -\ +$", the output symbol sequence is "$+\ \mathbf{1}\ \mathbf{1}\ -\ \mathbf{1}\ \mathbf{0}\ +$". The symbol sequence is then encoded into a bitstream by arithmetic coding (AC) [135].

3.2 Fast and PSNR-Improved WDR Algorithms

The major difference between the algorithms discussed below and the original WDR is in the encoding of symbols generated in the refinement pass. While Tian and Wells apply AC [135] to encode the refinement-pass symbols along with the sorting-pass symbols under the same context, we found that the encoding of the refinement-pass symbols should be separated from the encoding of the sorting-pass symbols to reach either a slightly higher compression ratio (with AC) within a similar time, or faster compression speed (without AC) with a comparable compression ratio.

In the WDR, the resultant output symbol sequence of the refinement pass is encoded together with the resultant symbol sequence of the sorting pass under the same context by AC. It means that we have only one alphabet A of four symbols, {**0**, **1**, +, −} for both passes. The AC adaptively updates the statistics of the symbols, *i.e.*, the occurrence frequency of the symbols, and determines the corresponding code length for each symbol.

In the proposed algorithm, dubbed modified WDR (MWDR), the refinement pass symbol sequence is separated from the sorting pass symbol sequence and is encoded under a different context that has an alphabet B of only two symbols, {**0**, **1**}.

From now on, we will call the symbols {**0**, **1**} *digit symbols* and the symbols {+, −} *sign symbols* for convenience.

Assume we have a symbol sequence of length N and n symbols ($n < N$) have already been encoded using the WDR. Applying the adaptive model in [135], we can see that the code length for the $(n+1)$-th symbol is determined by the occurrence frequency of the four entries in alphabet A after encoding the first n symbols, namely, $f_{n,0}$, $f_{n,1}$, $f_{n,+}$, and $f_{n,-}$. The estimated probability of the four symbols after n symbols have been encoded is:

$$\hat{p}_x = \frac{f_{n,x}}{\sum_x f_{n,x}},\ x \in A \tag{3.1}$$

In our proposed algorithm, there exist two contexts and we have new equations on the estimated probabilities.

$$\begin{cases} \hat{p}_{s,x} = \frac{f_{s,n,x}}{\sum_x f_{s,n,x}}, & x \in A \\ \hat{p}_{r,y} = \frac{f_{r,n,y}}{\sum_x f_{s,n,y}}, & y \in B \end{cases} \tag{3.2}$$

where $\hat{p}_{s,x}$ denotes the estimated probability of symbol x in the sorting pass and $\hat{p}_{r,y}$ denotes the estimated probability of symbol y in the refinement pass. The entries' occurrence frequency satisfies the following conditions:

$$f_{n,+} = f_{s,n,+} \tag{3.3}$$

$$f_{n,-} = f_{s,n,-} \tag{3.4}$$

$$f_{n,0} = f_{s,n,0} + f_{r,n,0} \tag{3.5}$$

$$f_{n,1} = f_{s,n,1} + f_{r,n,1} \tag{3.6}$$

where $f_{s,n,+}$ and $f_{s,n,-}$ denote the frequency of the sign symbols in the sorting pass when n symbols have been encoded. Similarly, $f_{s,n,0}$ and $f_{s,n,1}$ denote the frequency of the digit symbols in the sorting pass when n symbols have been encoded. $f_{r,n,0}$ and $f_{r,n,1}$ denote the frequency of the digit symbols in the refinement pass when n symbols have been encoded. Equation (3.3) and equation (3.4) are valid since there is no sign symbol output from the refinement pass. Equation (3.5) and equation (3.6) are valid because both the sorting pass and the refinement pass produces digit symbols.

According to our experiments as well as observation by other researchers [105], we have

$$\begin{cases} f_{r,n,0} \approx f_{r,n,1} \\ f_{n,+} \approx f_{n,-} \end{cases} \tag{3.7}$$

This is intuitive because, otherwise, we would have enjoyed a much higher compres-

sion gain. Equation (3.7) simplifies our analysis by assigning

$$\hat{p}_0 \approx \hat{p}_1 \approx p \tag{3.8}$$

$$\hat{p}_+ \approx \hat{p}_- \approx q \tag{3.9}$$

$$\hat{p}_{s,0} \approx \hat{p}_{s,1} \approx p_s \tag{3.10}$$

$$\hat{p}_{s,+} \approx \hat{p}_{s,-} \approx q_s \tag{3.11}$$

$$\hat{p}_{r,0} \approx \hat{p}_{r,1} \approx 0.5 \tag{3.12}$$

where $p + q = 0.5$ and $p_s + q_s = 0.5$. In the proposed MWDR, the expected code-length $E\{l_{n+1}\}$ of the $(n+1)$-th symbol is

$$E_{r,1}\{l_{n+1}\} = \hat{p}_{r,0} \log \frac{1}{\hat{p}_{r,0}} + \hat{p}_{r,1} \log \frac{1}{\hat{p}_{r,1}} \approx 1 \tag{3.13}$$

when the symbol is encoded in the refinement pass. From now on, we use log to represent $\log_2$ for simplicity. If the symbol is encoded by the original WDR algorithm, the expected code length is

$$E_{r,2}\{l_{n+1}\} = \frac{\hat{p}_0}{\hat{p}_0 + \hat{p}_1} \log \frac{1}{\hat{p}_0} + \frac{\hat{p}_1}{\hat{p}_0 + \hat{p}_1} \log \frac{1}{\hat{p}_1} \tag{3.14}$$

According to the log sum inequality [19],

$$\frac{\hat{p}_0}{\hat{p}_0 + \hat{p}_1} \log \hat{p}_0 + \frac{\hat{p}_1}{\hat{p}_0 + \hat{p}_1} \log \hat{p}_1 \geq \log \frac{\hat{p}_0 + \hat{p}_1}{2} \tag{3.15}$$

Hence, equation (3.14) turns into

$$E_{r,2}\{l_{n+1}\} \approx 1 - \log(\hat{p}_0 + \hat{p}_1) \approx \log \frac{1}{p} \tag{3.16}$$

according to equation (3.8). Consequently, the difference in code length upon using

the two algorithms becomes

$$\Delta l_{r,n+1} = E_{r,1}\{l_{n+1}\} - E_{r,2}\{l_{n+1}\} \approx 1 + \log p \leq 0 \tag{3.17}$$

because $p \leq 0.5$.

When the symbol is produced from the sorting pass, we have the following expected code-length if we use the MWDR:

$$E_{s,1}\{l_{n+1}\} = \sum_x \hat{p}_{s,x} \log \frac{1}{\hat{p}_{s,x}} \approx 2p_s \log \frac{1}{p_s} + 2q_s \log \frac{1}{q_s},\ x \in A \tag{3.18}$$

Similarly, we have the following expected code-length if we use the WDR:

$$E_{s,2}\{l_{n+1}\} = \sum_x \hat{p}_x \log \frac{1}{\hat{p}_x} \approx 2p \log \frac{1}{p} + 2q \log \frac{1}{q},\ x \in A \tag{3.19}$$

Now, the difference in the expected code-length upon using the two algorithms becomes

$$\begin{aligned} \Delta l_{s,n+1} &= E_{s,1}\{l_{n+1}\} - E_{s,2}\{l_{n+1}\} \\ &\approx 2p \log p + 2q \log q - 2p_s \log p_s - 2q_s \log q_s \end{aligned} \tag{3.20}$$

Based on the inequality

$$\frac{a}{b} \leq \frac{a+c}{b+c}, \quad \text{when } a < b \text{ and } a, b, c \in Z^+ \tag{3.21}$$

we have

$$\begin{aligned} p_s - q_s &\approx \frac{f_{s,n,0} - f_{s,n,+}}{f_{s,n,0} + f_{s,n,+}} \\ &= \frac{f_{s,n,0} - f_{n,+}}{f_{s,n,0} + f_{n,+}} \\ &\leq \frac{f_{s,n,0} - f_{n,+} + f_{r,n,0}}{f_{s,n,0} + f_{n,+} + f_{r,n,0}} \\ &= \frac{f_{n,0} - f_{n,+}}{f_{n,0} + f_{n,+}} \\ &\approx p - q \end{aligned} \tag{3.22}$$

because in general the probability of the digital symbols ("**0**" and "**1**") is higher than the probability of the sign symbols, ("+" and "−"). Based on equation (3.22), $p + q \approx 0.5$, and $p_s + q_s \approx 0.5$, we have $\Delta l_{s,n+1} \geq 0$. Now it is clear that although $E\{l_{n+1}\}$ is decreased in the refinement pass, it is increased in the sorting pass. It is necessary to show that, on average, the decrease in the expected code-length in the refinement pass surpasses the increase in the expected code-length in the sorting pass.

$$\begin{aligned} \Delta l_{n+1} &= \Delta l_{r,n+1} + \Delta l_{s,n+1} \\ &\approx 1 + \log p + 2p \log p + 2q \log q - 2p_s \log p_s - 2q_s \log q_s \\ &= 1 + \log p + 2p \log p + 2(0.5 - p) \log(0.5 - p) \\ &\quad -2p_s \log p_s - 2(0.5 - p_s) \log(0.5 - p_s) \end{aligned} \tag{3.23}$$

where the relationship between p and p_s satisfies the following conditions:

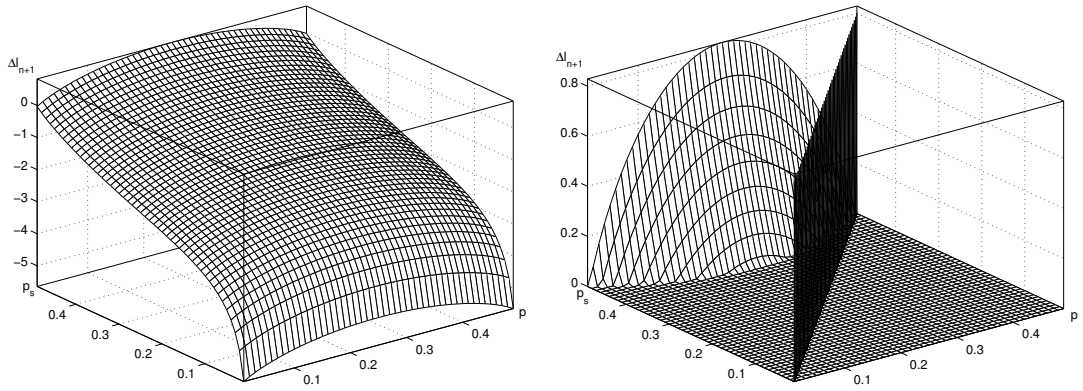
$$\begin{aligned} p_s &\approx \frac{f_{s,n,0}}{f_{s,n,0} + f_{n,+}} \\ &\leq \frac{f_{s,n,0} + f_{r,n,0}}{f_{s,n,0} + f_{n,+} + f_{r,n,0}} \\ &= \frac{f_{n,0}}{f_{n,0} + f_{n,+}} \\ &\approx p \end{aligned} \tag{3.24}$$

Fig. 3.1 shows that when equation (3.24) is satisfied, the decrease of expected code-length in the refinement pass always exceeds the increase of expected code-length in the sorting pass, *i.e.*, $\Delta l_{n+1} < 0$. It is also shown that when p decreases, which is true when the encoding bitrate becomes higher, $|\Delta l_{n+1}|$ becomes larger. This implies that more bit budget can be saved by the MWDR when the bitrate gets higher.

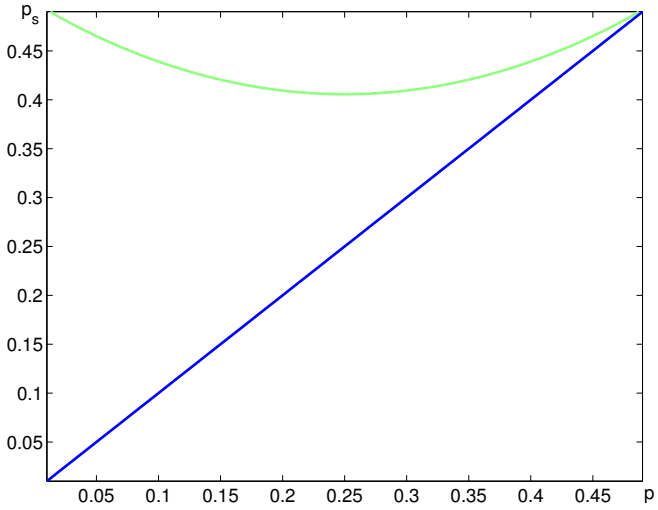
With the analysis above, we can adopt two different strategies to modify the WDR. If we need higher compression ratio, we can apply AC encoding to the sorting pass symbol sequence with the four-symbol context and the refinement pass symbol sequence with the two-symbol context. We call this algorithm *MWDR-A*. If we want reduced computational complexity, we can simply use two bits to represent each of the four symbols in the sorting pass and output the digit symbols without any further processing in the refinement pass. This algorithm is called *MWDR-B*. The detailed steps of these two algorithm are presented in the following:

1. **Initialization:** output $l = \lfloor \log_2(\max_{(i,j)}\{|c_{i,j}|\}) \rfloor$; set threshold $T = 2^l$; set the index of last significant coefficient n to 0; initialize the SCS and the TPS as empty lists; add all wavelet coefficients to the ICS according to the predefined scan-order in the WDR.

2. Store the indices of the coefficients in the ICS. After the initialization step, coefficient $c_{i,j}$ in the wavelet pyramid is labeled as c_k in the ICS, where $k \in$



(a) (b)



(c)

Figure 3.1: (a) Plot of Δl_{n+1} versus p and p_s. (b) An enlarged portion of (a), where $\Delta l_{n+1} > 0$. (c) The contour plot of $\Delta l_{n+1} = 0$ and $p = p_s$. It is shown that $\Delta l_{n+1} < 0$ when $p_s < p$.

$[1, N]$, $k \in Z$. N denotes the total number of coefficients in the wavelet pyramid.

3. **Sorting Pass:** for each entry c_k in the ICS do: if $|c_k| \geq T$,

   (a) move c_k to the TPS, output the reduced binary representation of $k - n$, output the sign of c_k;

   (b) set $n = k$;

   When the end of the ICS is reached, the reduced binary representation of $N + 1 - k$ and a "+" sign are outputted to demarcate the end of current bit-plane. By doing this, the last index for significant coefficients is always $N + 1$ in each bit-plane. This index also serves as an error-detection marker in the decoding process.

4. **Refinement Pass:** for each entry in the SCS, output its l-th most significant bit.

5. **Sets Update:** append the TPS to the end of SCS; assign a new index for each entry in the shrunken ICS.

6. **Quantization-Step Update:** decrement l by 1, divide T by 2, and go back to Step 3.

Here, an encoding example is given to clarify better understanding of the WDR algorithm. A two-level wavelet pyramid of size 4×4 plus the index for each coefficient is given in Fig. 3.2. The scan-order follows the the index increment. It is shown that the scan-order goes column-by-column in the HL subbands, which is different from the scan-order of subbands of the other three orientations. In this example, we assign these four code-words for the four symbols, 00 for **"0"**, 01 for **"1"**, 10 for "−", and 11 for "+" in MWDR-B. Using the algorithm listed above, we can encode this pyramid by the following steps:

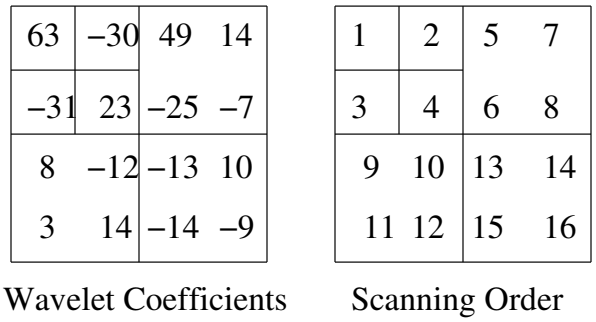


Figure 3.2: An example two-level wavelet pyramid of size 4×4. The scanning order is illustrated by the corresponding index for each coefficient.

1. **Initialization:** Since the the largest magnitude is 63, we output $l = 5$; set threshold T to $2^5 = 32$ for the highest bit-plane; set the index of the last significant coefficient n to 0; initialize the SCS and the TPS as empty lists; add all wavelet coefficients to the ICS according to the predefined scan-order; store the indices of coefficients in the ICS; the total number of entries in the ICS is $N = 16$.

2. **Sorting Pass:** Compare the magnitude of each coefficient with 32, we have two significant coefficients in this bit-plane, the 1st and the 5th. So we need to output two numbers, 1 and 4. Respectively, their reduced binary representations are "null" and **"0 0"**. We also need to output the reduced binary representation of $16 + 1 - 5 = 12$, *i.e.* **"1 0 0"** and a "+" at the end of this bit-plane. So the output symbol sequence is: **"+ 0 0 + 1 0 0 +"**. If the MWDR-A is employed, the symbol sequence will be encoded under the four-symbol context using an AC. If the MWDR-B is used, the output bitstream is already known: 11 00 00 11 01 00 00 11. Note that 63 and 49 are to be moved to the TPS.

3. **Refinement Pass:** The SCS is currently empty, so nothing to be done here.

4. **Sets Update:** Append the TPS to the end of the SCS; assign new indices for

the remaining fourteen entries in the ICS.

5. **Quantization-Step Update:** set $l = 4$ and $T = 16$.

6. **Sorting Pass:** Now the ordering of entries in the ICS is: $\{-30_1,\ -31_2,\ 23_3,\ -25_4,\ 14_5,\ -7_6,\ 8_7,\ -12_8,\ 3_9,\ 14_{10},\ -13_{11},\ 10_{12},\ -14_{13},\ -9_{14}\}$. Comparing entries in the ICS with $T = 16$, we have four significant coefficients, the 1st, the 2nd, the 3rd, and the 4th. Based on the same procedures above, we have an output symbol sequence: "$-\ -\ +\ -$ **1 0 1** $+$". If the MWDR-B is used, the output bitstream is: 10 10 11 10 01 00 01 11. Move 30, 31, 23, and 25 to the TPS. Only the magnitudes of the coefficients are moved to the TPS since the sign information has been encoded.

7. **Refinement Pass:** Since $(63)_{10} = (111111)_2$ and $(49)_{10} = (110001)_2$, the bits on the 4th bit-plane of these two coefficients are "**1**" and "**1**", respectively. If we use the MWDR-A, these two symbols will be encoded under the two-symbol context. If the MWDR-B is used, we simply output two bits, 11.

8. **Sets Update:** Append the TPS to the end of the SCS; assign new indices for the remaining ten entries in the ICS.

9. **Quantization-Step Update:** set $l = 3$ and $T = 8$, and go back to Step 2.

Although only the encoding of two most significant bit-planes is listed in the above description, it is not difficult to extend the procedures to other lower bit-planes.

The proposed techniques were evaluated using four 512×512, 8 bpp standard gray-scale images, *Lena*, *Goldhill*, *Boat*, and *Barbara*, all can be downloaded from `http://links.uwaterloo.ca/bragzone.base.html`. A lifting implementation [22] of the bi-orthogonal wavelet filter bank proposed by Cohen, Daubechies and Feauveau [18] was used with a decomposition level of 6. The programs ran on a Linux server with

a Pentium-III 500-MHz CPU with 256 MB of memory. The Linux kernel version was 2.4.7 and the GNU C Compiler (GCC) version was 2.96.

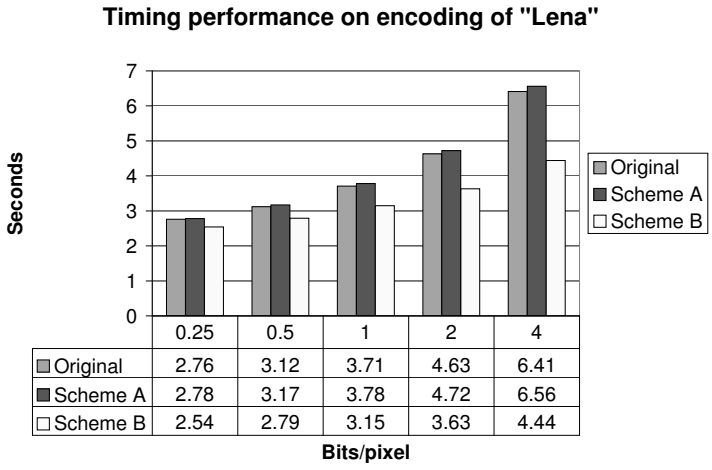
The compression performance of the three algorithms is compared in terms of the peak signal-to-noise ratio (PSNR)

$$\text{PSNR} = 10 \log_{10} \left(\frac{255^2}{\text{MSE}} \right) \text{ dB} \tag{3.25}$$

where MSE is the mean square error between the original image and the decoded image. All bitrates given are the exact output bitrates, including bits spent on side information such as image size and decomposition level. The timing performance of the three algorithms is measured by the *clock*() function offered in the GNU ANSI C library. Since the *clock*() function only returns an approximation of the processor time used by the program, we ran the program thirty times in succession, and calculated the average of the processor time used by the programs. Although we took this step to avoid large timing error, the bar-graph plotted in Fig. 3.3 should only be interpreted as a qualitative comparison among the algorithms.

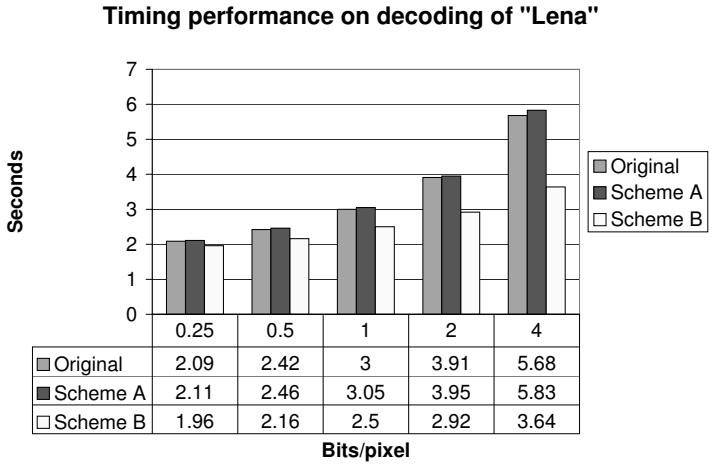
It is shown in Fig. 3.3 that MWDR-B always consumed the least time on encoding and decoding. This results from the simplicity in the mapping from symbol stream to bitstream. Another observation is that the timing gap enlarges with the increase of the coding bitrate between MWDR-B and the other two algorithms. This can be explained by the fact that entropy-encoding longer output symbol sequences needs more time. When a near-lossless bitrate is reached, it can be seen that about 1/3 of both the encoding and the decoding time can be saved without sacrificing any compression performance when compared to the WDR algorithm. The PSNR difference among the three algorithms is small enough that no difference can be visually perceived.

The compression performance of the three algorithms on four different test images



	0.25	0.5	1	2	4
Original	2.76	3.12	3.71	4.63	6.41
Scheme A	2.78	3.17	3.78	4.72	6.56
Scheme B	2.54	2.79	3.15	3.63	4.44

(a)



	0.25	0.5	1	2	4
Original	2.09	2.42	3	3.91	5.68
Scheme A	2.11	2.46	3.05	3.95	5.83
Scheme B	1.96	2.16	2.5	2.92	3.64

(b)

Figure 3.3: The comparison of timing performance on the coding of the *Lena* image (512×512, 8 bpp): (a) encoding of *Lena* at various bitrates; (b) decoding of *Lena* at various bitrates.

Table 3.1: The PSNR performance of the algorithms on the compression of *Lena*.

	bitrate (bpp)				
	0.15	0.25	0.5	1	2
Original WDR, PSNR (dB)	31.16	33.22	36.29	39.47	43.88
Scheme A, PSNR (dB)	31.34	33.38	36.50	39.66	44.24
Scheme B, PSNR (dB)	31.12	33.24	36.36	39.54	44.02

are compared in tables 3.1-3.4. It can be seen that the MWDR-A always achieves the best compression performance by encoding the output symbol sequence from the sorting pass and the refinement pass separately under different contexts. The price paid for doing this is just a little more running time compared to the WDR algorithm. On a more interesting comparison between the WDR algorithm and the MWDR-B, it is found that at a certain breakpoint, the performance of the MWDR-B may exceed that of the original WDR algorithm. This is due to two reasons. First, the MWDR-B saves bit budget by encoding of a refinement pass symbol with only one bit, while on average the WDR spends more than one bit for each of those symbols. Second, by excluding the **"0"** and **"1"** symbols of the refinement pass from being arithmetic-encoded together with those symbols of the sorting pass, the "skewness" of the symbol probability distribution in the sorting pass is decreased, which should have caused AC to be less efficient if we employ the AC in the MWDR-B. This justifies the exclusion of the AC from the MWDR-B. Since the PSNR difference among the three algorithms is small enough, it is the author's opinion that the MWDR-B should be the optimal algorithm used in practice.

3.3 Context Modeling WDR

The exclusion of the zerotree data structure from the WDR algorithm makes it simpler than zerotree-based algorithms, but a by-product of this coding strategy is

Table 3.2: The PSNR performance of the algorithms on the compression of *Goldhill*.

	bitrate (bpp)				
	0.15	0.25	0.5	1	2
Original WDR, PSNR (dB)	28.65	30.24	32.66	35.82	40.73
Scheme A, PSNR (dB)	28.72	30.36	32.80	36.04	41.18
Scheme B, PSNR (dB)	28.64	30.25	32.61	35.86	40.85

Table 3.3: The PSNR performance of the algorithms on the compression of *Boat*.

	bitrate (bpp)				
	0.15	0.25	0.5	1	2
Original WDR, PSNR (dB)	28.32	30.47	33.57	38.16	43.63
Scheme A, PSNR (dB)	28.42	30.57	33.82	38.44	43.99
Scheme B, PSNR (dB)	28.32	30.34	33.55	38.23	43.85

Table 3.4: The PSNR performance of the algorithms on the compression of *Barbara*.

	bitrate (bpp)				
	0.15	0.25	0.5	1	2
Original WDR, PSNR (dB)	25.79	27.25	31.12	35.86	42.27
Scheme A, PSNR (dB)	25.85	27.32	31.35	36.25	42.83
Scheme B, PSNR (dB)	25.67	27.23	31.20	36.05	42.61

that no effort is made to exploit the statistical dependency across scales. Later, Walker [132] observed this drawback and incorporated the parent-children relationship into the original WDR to achieve better compression. By visual inspection on Fig. 2.1 we can see clearly that wavelet coefficients in the same orientation across scales are not statistically independent. Large magnitude coefficients tend to occur at neighboring spatial locations, and also at the same relative spatial locations of subbands at adjacent scales and orientations [9].

Researchers have been proposing probability models for wavelet coefficients conditioned on their "parents" and "neighbors", which function as the *contexts* of the wavelet coefficients. Quantization and compression of a coefficient is achieved by an optimal scalar quantizer and an adaptive AC afterwards based on the probability model [9, 14, 64, 139]. It is necessary to point out here that the implementation of these traditional context modeling methods is hampered by the complicated design of their optimal quantizers and relative complexity of context-switched adaptive AC.

Ordentlich *et al.* [86] proposed a very simple and efficient context modeling approach that classifies one bit-plane into several sequences (fractional bit-planes) based on already coded bit-planes.

In the following section, we will show the simplicity of the WDR and how the context modeling approach facilitates better compression. Suppose we have an image of size $M \times N$. The total number of coefficients in the wavelet pyramid is MN. Each coefficient in the pyramid is assigned an index according to the scan-order as in Fig. 3.2. The highest bit-plane, or the initial bit-plane, $B_{p_{\max}}$ is determined by the coefficients with the largest magnitude

$$B_{p_{\max}} = \lfloor \log_2 \max\{|c|\} \rfloor \tag{3.26}$$

where $\lfloor x \rfloor$ denotes the largest integer that is not greater than x. For each coefficient c

in bit-plane B_p that is insignificant before B_p is encoded, we compare its magnitude against threshold $T_p = 2^p$. If $|c| \geq T_p$, we say that c is a significant coefficient. Suppose c is the n-th significant coefficient in bit-plane B_p. The difference (run-length) between its index and the index of the $(n-1)$-th significant coefficient can be expressed as

$$l_{p,n} = i_{p,n} - i_{p,n-1} \tag{3.27}$$

where $i_{p,n}$ denotes the location index for the n-th significant coefficient in bit-plane B_p and $i_{p,0} = 0$. Hence, the number of generated output symbols by the binary reduction method can be expressed as

$$\lceil \log_2 l_{p,n} \rceil - 1 \tag{3.28}$$

These run length outputs are delimited by the sign symbols. Therefore, run length output symbol(s) for each significant coefficient will be followed by its sign symbol as delimiter. To mark the end of one bit-plane, we have a virtual significant coefficient with assigned location index being $(MN+1)$ and sign symbol being "+".

Assume that the number of significant coefficients in bit-plane B_p is S_p, the total number of significant coefficients we need to encode is $S_p + 1$ because we have a virtual significant coefficient as the bit-plane end-marker. The sum of the run length of all the significant coefficients satisfies

$$\sum_{i=1}^{S_p+1} l_{p,i} = MN + 1 \tag{3.29}$$

To make the analysis simple, we assume that no entropy coder is used. Each of the four symbols is encoded using two bits. Now we will have the rate information

regarding the encoded bit-planes.

$$R_p = \sum_{i=p}^{p_{max}} \left(\underbrace{2 \cdot \sum_{j=1}^{S_i+1} (\lceil \log_2 l_{i,j} \rceil - 1)}_{\text{run length symbols}} + \underbrace{2 \cdot (S_i + 1)}_{\text{sign symbols}} + \underbrace{\sum_{k=i+1}^{p_{max}} S_k}_{\text{refinement symbols}} \right) \tag{3.30}$$

where R_p denotes the rate when encoding the bit-planes from $B_{p_{\max}}$ to B_p. From (3.30), it is clear that compression is achieved by RLC of indices of significant coefficients at each bit-plane. With (3.30), we can see that the worst case will be that significant coefficients distribute uniformly along the the path defined by scan-order. By changing the scan-order, we can change the distribution of significant coefficients, and thus change the efficiency of the RLC. Now the choice of scan-order becomes the key factor for a better compression.

The ideal case for the binary RLC occurs when the indices of all significant coefficients are consecutive without the interruption of insignificant coefficients. For example, if we have S_p significant coefficients in bit-plane B_p, the best compression can be achieved if the first S_p coefficients are all significant, where only $(\lceil \log_2(MN + 1 - S_p) \rceil - 1)$ symbols will be output as run-length symbols. The WDR uses a fixed scan-order, which is based on the well-known fact that coefficients with large magnitudes tend to appear in the low-pass subbands and coarser scales. Hence, the scanning procedure starts from the coarsest scale and ends at the finest scale. The low-pass subbands are scanned before the high-pass subbands at each scale.

With this scan-order, we can see that in general $l_{p,n} \leq l_{p,n+1}$ because significant coefficients tend to be denser in the front of a scanning path. The idea of the proposed scheme, which we named context modeling WDR (CM-WDR), is to move a potentially significant coefficient at the back of the scanning path forward to the front of the scan-order based on the significance of its parent and neighbors.

By moving a potential significant coefficient forward to the front of the scan-

order, we have a hit-or-miss situation. If this coefficient turns out to be significant and it is the n-th significant coefficient in original fixed order, the saving in symbol output is

$$\begin{aligned}\Delta &= \lceil \log_2 l_{p,n} \rceil - 1 + \lceil \log_2 l_{p,n+1} \rceil - 1 - \\ &\quad (\lceil \log_2 (l_{p,n} + l_{p,n+1} - 1) \rceil - 1) \end{aligned} \tag{3.31}$$

By assuming $l_{p,n} \approx {}_{p,n+1}$, we can save about $(\lceil \log_2 l_{p,n} \rceil - 2)$ symbols.

If the moved coefficient turns out to be insignificant, we are penalized by this miss. The worst case is that we output one more symbol when we insert this insignificant coefficients between two consecutive significant coefficients. Evaluating the penalty and the benefit, there is a better compression performance achieved in general with the context modeling approach.

In the following list, we give a detailed description of the processing steps of the CM-WDR algorithm before further discussion.

1. *Transformation:* Perform 2-D DWT on the original image. In the experiments described below, a lifting implementation [22] of the bi-orthogonal wavelet filter-bank proposed by Cohen, Daubechies and Feauveau [18] has been used with a decomposition level of 6.

2. *Context setting:* Let i denote a coefficient in the decomposed wavelet pyramid. We have $P(i)$ being the parent of i in terms of the usual parent-children relationship [108]. In addition, we let $N(i)$ denote a to-be-specified collection of spatially contiguous neighbors of i from the same subband as i.

3. *Scanning order setting:* Each subband at level l is partitioned into macro-blocks of size $2^{7-l} \times 2^{7-l}$. Within each block, the index of the ith coefficient in the scan-order are obtained by de-interleaving the odd and the even bits of i.

This scan-order within each subband is different from the WDR [121] while the scanning priority of orientations and levels is identical to that of the WDR.

4. *Initialization:* Determine the initial threshold T using the same procedure as the WDR. Also, initialize the indices of all coefficients in the wavelet pyramid according to the scan-order.

5. *Sorting pass:* Record positions for the *new* significant coefficients, *i.e.*, the indices for those whose magnitudes are greater than or equal to the *current* threshold. Encode these positions with the RLC method as in the WDR.

6. *Refinement pass:* Record the *refinement bits*, *i.e.*, the binary representations of the *old* significant coefficients, which fall into the bit-plane defined by current threshold. These refinement bits can be left without further encoding.

7. *Index update:* An updated index array is generated based on the remaining insignificant coefficients. The updated index array consists of three sub-arrays. The first sub-array is called the *significant neighbor* sub-array (SNS), which contains all of the insignificant coefficients that have at least one significant neighbor. The second sub-array is called the *significant parent* sub-array (SPS), which contains all insignificant coefficients whose parent is significant. The third sub-array is called the *run* sub-array (RS), which contains the remaining insignificant coefficients which do not satisfy the above conditions. Note that each insignificant coefficient can appear in only one sub-array. The priority of the SNS is higher than the SPS, *i.e.*, a coefficient is first checked to see whether it has significant neighbor(s). Three sub-arrays are concatenated to generate the updated index array. The concatenation order is: SNS + SPS + RS.

8. *Threshold update:* Divide the current threshold by 2, and repeat from the sorting pass until a given bit budget is reached.

In the decoder, the steps above are recapitulated to produce a quantized output. The final reconstruction value is set to the mid-point of the quantization bin that it lies in.

The reason that we put the SNS in the first part of the updated index array is that we expect *a priori* that *new* significant coefficients in the SNS will be identified with higher probability in the next round of testing than those of the SPS. This agrees with the observation in [86].

Another important feature of the proposed algorithm is that the AC is avoided, which simplifies the coding architecture. Each of the four symbols generated in the sorting pass is encoded by two bits. However, this is not to say that the AC cannot be applied to the CM-WDR, we would like to claim that the additional computational complexity brought by the AC is not justified by the reduced reconstruction distortion.

A natural extension of the CM-WDR is to incorporate an AC stage into the encoding of symbol sequences from sorting passes and refinement passes. Based on the analysis in §3.2 on the statistics of the produced symbols, two contexts are employed in the AC as proposed in the MWDR-A. We call this AC variant of the context modeling scheme CM-WDR/AC.

Table 3.5 shows the PSNR, as defined in equation (3.25), results for the CM-WDR and the CM-WDR/AC on some 8 bpp gray-scale still images. All the images have size of 512×512. The five rows of the PSNR values for each image corresponds to the CM-WDR, the CM-WDR/AC, the WDR, the SPIHT (without arithmetic coding), and the SPIHT-AC (with arithmetic coding). The results for the SPIHT and the SPIHT-AC were obtained using the implementation in QccPack [28].

As can be seen, the CM-WDR rarely performs worse than 0.1 dB below the SPIHT-AC at low bitrates, and always performs better than the SPIHT and the WDR at all bitrates. The performance gap between the WDR and the CM-WDR

Table 3.5: PSNR (dB) vs. bitrate (bpp) for SPIHT, SPIHT-AC, WDR, and CM-WDR.

Image		Output bit rate (bpp)			
		0.15	0.25	0.5	1.0
Lena	SPIHT	31.33	33.62	36.74	39.90
	SPIHT-AC	31.71	34.00	37.11	40.29
	WDR	31.16	33.22	36.29	39.47
	CM-WDR	31.61	33.89	36.99	40.05
	CM-WDR/AC	31.72	34.01	37.10	40.19
Goldhill	SPIHT	28.59	30.16	32.57	35.86
	SPIHT-AC	28.89	30.44	32.96	36.38
	WDR	28.65	30.24	32.66	35.82
	CM-WDR	28.78	30.27	32.78	36.09
	CM-WDR/AC	28.87	30.39	32.88	36.24
Boat	SPIHT	28.37	30.34	33.74	38.34
	SPIHT-AC	28.63	30.74	34.20	38.91
	WDR	28.32	30.47	33.57	38.16
	CM-WDR	28.50	30.51	33.93	38.49
	CM-WDR/AC	28.63	30.71	34.12	38.75
Barbara	SPIHT	25.50	27.60	31.60	36.79
	SPIHT-AC	26.08	28.20	32.20	37.42
	WDR	25.79	27.25	31.12	35.86
	CM-WDR	26.07	28.20	32.15	37.31
	CM-WDR/AC	26.15	28.25	32.28	37.42

ranges from 0.1 dB to 1.5 dB at various bitrates on the tested images. It is also observed that the WDR performs nearly as well as the SPIHT at low bitrates. It is also shown that the performance of the CM-WDR/AC ties with that of the SPIHT-AC.

Table 3.6 compares the performance of the CM-WDR and the CM-WDR/AC with that of the WDR from a different perspective. Here, the total number of significant coefficients identified is used as a criterion. For each encoded bit-plane, both the CM-WDR and the WDR spend the same bit budget on the the refinement pass. Since the adaptive scan-order in the CM-WDR increases the *skewness* of the distribution of the significant coefficient in the to-be-scanned array, the efficiency of

Table 3.6: Comparison on the number of encoded significant coefficients for the CM-WDR, the CM-WDR/AC, and the WDR.

Image		Output bit rate (bpp)			
		0.15	0.25	0.5	1.0
Lena	CM-WDR	6928	12058	24292	48376
	CM-WDR/AC	7091	12378	24934	50015
	WDR	6351	10124	20285	42008
Goldhill	CM-WDR	7144	10488	23636	50274
	CM-WDR/AC	7351	11210	24150	51483
	WDR	6872	10335	23291	48508
Boat	CM-WDR	6801	11060	23381	46739
	CM-WDR/AC	7109	11151	24021	46739
	WDR	6427	10990	22372	46151
Barbara	CM-WDR	7118	13981	27838	53990
	CM-WDR/AC	7425	14147	28394	54743
	WDR	6936	11603	23421	45589

the RLC is higher. The bit budget saved in the sorting pass allows the CM-WDR to encode more significant coefficients. The AC stage in the CM-WDR/AC scheme allows it to encode even more coefficients.

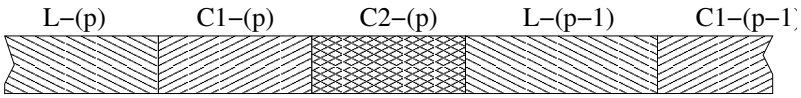
3.4 Color Image Coding with Context Modeling

Up to now, the basic assumption in our discussion is that a raw image is a gray-scale (or monochrome) image. However, color images are almost always involved for real-world applications. A direct extension of the proposed schemes to color image coding is to encode the components of a color image as three independent gray-scale images. This strategy is adopted by the latest still image coding standard JPEG-2000 [71]. In JPEG-2000, an image that has multiple components, *e.g.*, an RGB image, is first de-correlated using a color transform before applying the wavelet transform. The de-correlated components are treated independently as gray-scale images after the color transformation.

However, two questions arise with this simple strategy. First, the statistical dependency that exists among the de-correlated components is not exploited. It has long been observed that in color images the locations of significant spatial changes in the chrominance components are generally coincident with significant spatial changes in the luminance component [81]. Hence, the prediction of a chrominance sample can be made efficient by using the previously transmitted chrominance and luminance samples, and the present luminance sample. Second, explicit rate allocation among the color components may be needed, which complicates the task of rate control.

In order to utilize the statistical dependency among de-correlated components and retain the embeddedness and implicit bit allocation of wavelet coders, some researchers have proposed algorithms for embedded color image and video coding system. The extension of the EZW to color image coding is proposed as CEZW [109], where the zerotree data structure is modified to accommodate chrominance components. In the CEZW, each node in chrominance component has two parent nodes, one in the same chrominance component and the other in the luminance component. Hence, for each zerotree root in the luminance pyramid, all of its children, including those in chrominance components, are tested for their significance. In the extension of the SPIHT to color image coding [53], the list-initialization processes for the three color components are interleaved. However, this extension of the SPIHT to color image coding does not address the exploitation of the statistical dependency among the color components.

A novel wavelet coding algorithm, named color wavelet difference reduction (CWDR), for encoding color image is proposed in this section. The CWDR first transforms a color image into the YUV color space, and then applies the DWT to the Y, U, and V components. The encoding of the Y component is done by the CM-WDR discussed in §3.3. The encoding of the U and V components is based on the contextual information formed during the encoding of the Y component.



L–(p): The p–th bit–plane of luminance component
CX–(p): The p–th bit–plane of the X–th chrominance component

Figure 3.4: The composition of the output bit-stream.

3.4.1 Color Wavelet Difference Reduction

CWDR is based on similar context modeling approach adopted in §3.3. However, the problem of rate allocation persists before a method is found to assign bit budget among the three components. To retain the feature of embeddedness, we progressively interleave the bit-streams for the bit-planes of the three components as shown in Fig. 3.4. The order of the chrominance components is determined by their most significant bit-planes. The chrominance component with the higher most significant bit-plane is encoded before the other. If the most significant bit-planes of both chrominance components are equally high, U is encoded before V.

Each bit-plane of the luminance component (Y) is encoded using CM-WDR. During the encoding of the p-th bit-plane of Y, the contextual information for a node is formed by examining its parent node and neighboring nodes in the higher bit-planes as illustrated in Fig. 3.5. However, for bit-planes of the chrominance components (U and V), the contextual information for a peer node is formed by examining the corresponding node at the same coordinates of the same bit-plane of Y. This simplifies the context-modeling process and exploits the latest updated information, *i.e.*, the B_p of Y is used jointly with bit-planes higher than B_p. This is also the reason that we prefer interleaving the three components at the bit-plane level over interleaving at the coefficient level [53].

The efficacy of the context modeling method is demonstrated in Fig. 3.6. The white pels in Fig. 3.6 (a) and Fig. 3.6 (c) represent the significant coefficients of the

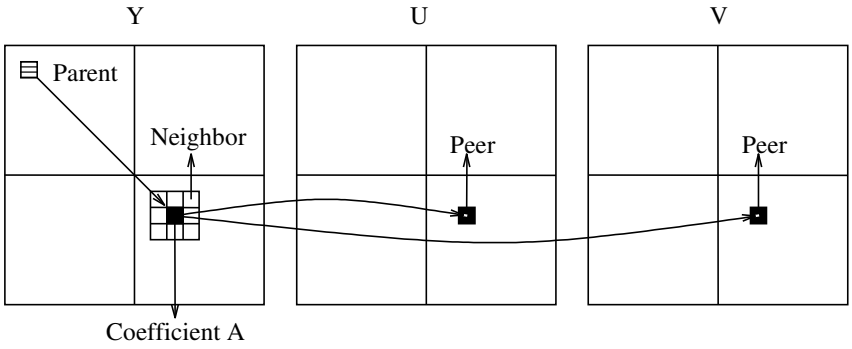


Figure 3.5: Coefficient A in Y is modeled by its parent and neighbors whilst A's peer coefficients in U and V are directly modeled by A.

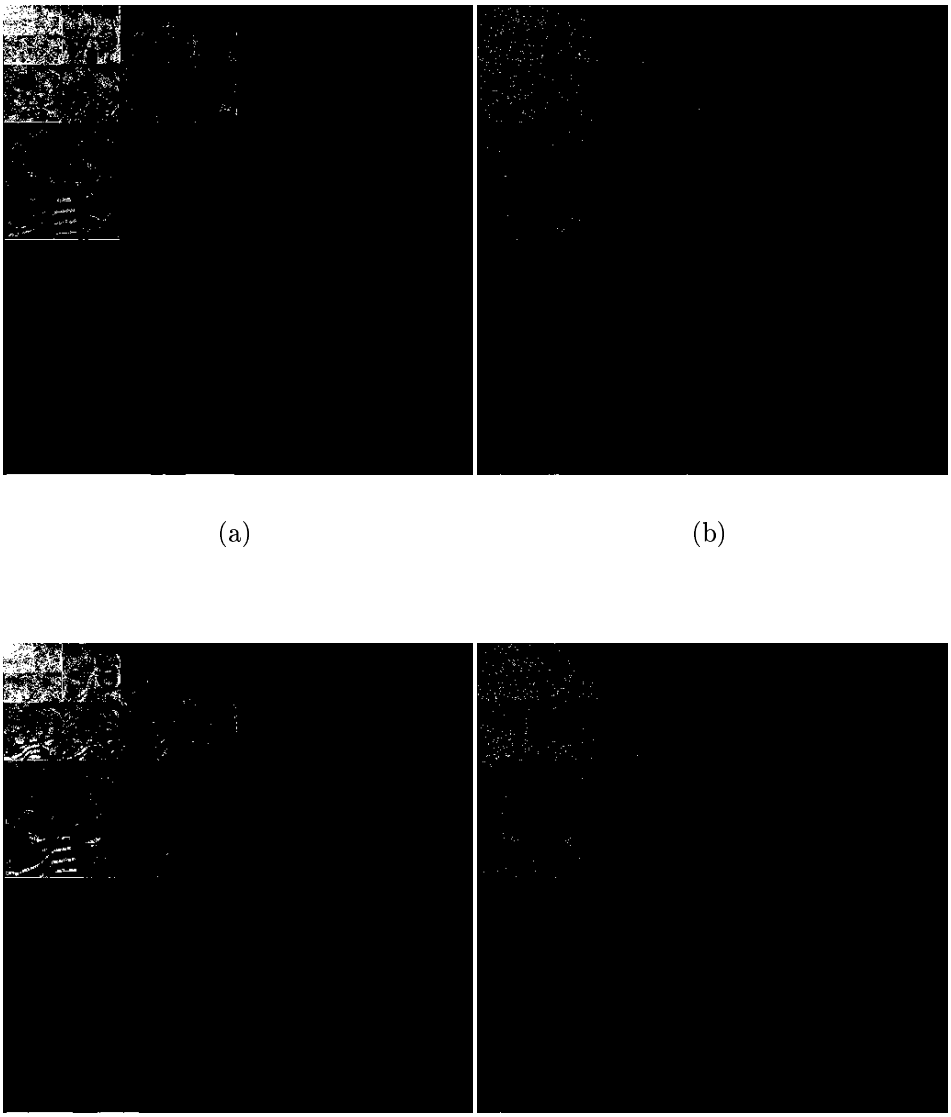
U and the V components at B_2 (with threshold value of 4). White pels in Fig. 3.6 (b) and Fig. 3.6 (d) represent those of the U and the V component whose significance fails to be predicted by using the contextual information from the Y component. It is clear that the simple context modeling approach for chrominance components are very efficient.

The pseudo-code for CWDR is listed in Algorithm 1:

Algorithm 1 The CWDR encoder.

for i := Y to V do {
  P_i := dwt_2d(i);
  $B_{i,\max} := \lfloor \log_2 \max(|P_i|) \rfloor$;
}
$B_{\max} := \max B_{i,\max}$;
if($B_{U,\max} < B_{V,\max}$) {encoding_order := Y $\rightarrow$ V $\rightarrow$ U;}
else {encoding_order := Y $\rightarrow$ U $\rightarrow$ V;}
$T := 2^{B_{\max}}$; $b := B_{\max}$;
while($r < R_{budget}$) do {
  for i := Y to V do {
    if($b > B_{i,\max}$) continue;
    if($i \neq$Y) {update_UV_scanning_order(P_i, b);
    sorting_pass(P_i, T);
    refinement_pass(P_i, T);
    if($i ==$ Y) update_scanning_order(P_i, b);
  }
  $T := T/2$; $b := b - 1$;
}

Note that the sorting pass and the refinement pass of the CWDR are the same as



(a) (b)

(c) (d)

Figure 3.6: Prediction efficacy of the context-model used on *Girl* at a decomposition level of 6: (a) the significant coefficients (white pels) at the B_2 of U, (b) the significant coefficients in (a) that fail to be identified by context-modeling, (c) the significant coefficients at the B_2 of V, (b) the significant coefficients in (c) that fail to be identified by context-modeling.

in §3.3. The major difference is in the scan-order updating procedure for chrominance components. When updating the scanning order for U (or V), the nodes whose corresponding peer nodes in Y are significant are scanned first. Since this approach is very efficient, the parent-children relationship and neighborhood relationship are not used for U and V, and the complexity of context modeling is further decreased.

3.4.2 Experimental Results

In our experiments, three RGB color images of size 512×512, *Girls*, *Lena*, and *Peppers*, downloaded from `http://links.uwaterloo.ca/colorset.base.html`, have been used. The CEZW and the JPEG-2000 algorithms are used as benchmark schemes. Instead of implementing CEZW, we directly downloaded the decoded copies of *Girls* from the authors' public FTP directory `ftp://skynet.ecn.purdue.edu/pub/dist/delp/icip97-coding`. For JPEG-2000, we have used the Kakadu implementation of the standard [117]. To evaluate the fidelity of the decoded images, both the color SNR that is based on the 1964 CIE formula [47] and the PSNR for each component are calculated. The color of pels is represented in CIE modified UCS color space (U^*, V^*, W^*). The color SNR is calculated as follows:

$$\text{SNR} = 10 \log_{10} \frac{(\Delta s_0)^2}{(\Delta s)^2} \tag{3.32}$$

where $(\Delta s)^2$ is the Euclidean distance between the original image and the decoded image. Note that $(\Delta s_0)^2$ is the Euclidean distance between the original image and its mean color:

$$(\Delta s)^2 = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (\Delta u_{ij}^*)^2 + (\Delta v_{ij}^*)^2 + (\Delta w_{ij}^*)^2 \tag{3.33}$$

and

Table 3.7: Color SNR (dB) and component PSNR (dB) versus bitrate for CEZW, JPEG-2000, and CWDR on *Girls*.

Bitrate (bpp)		color SNR	Y PSNR	U PSNR	V PSNR
0.5	CEZW	12.19	33.90	39.03	36.97
	JPEG2K	15.37	34.00	43.90	41.70
	CWDR	16.04	34.64	44.61	42.58
1.0	CEZW	15.17	37.38	41.88	39.64
	JPEG2K	18.20	37.09	46.54	43.92
	CWDR	17.46	38.64	44.61	43.15
1.5	CEZW	15.59	40.70	41.88	39.64
	JPEG2K	19.89	38.89	48.08	45.45
	CWDR	20.08	41.20	47.17	45.66

$$(\Delta s_0)^2 = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(u_{ij}^* - u_0)^2 + (v_{ij}^* - v_0)^2 + (w_{ij}^* - w_0)^2 \tag{3.34}$$

where

$$u_0 = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(u_{ij}^*)^2 \tag{3.35}$$

$$v_0 = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(v_{ij}^*)^2 \tag{3.36}$$

$$w_0 = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(w_{ij}^*)^2 \tag{3.37}$$

where M and N are the width and the height of an image, respectively.

As shown in Table 3.7, the color SNR gap between the CEZW and the CWDR is so significant that the CWDR at 0.5 bpp is better than the CEZW at 1.5 bpp. However, that gap between the JPEG-2000 and the CWDR is much smaller.

Since only the results for *Girls* are available for the CEZW, we compared the JPEG-2000 and the CWDR on the other two images. We can see that the CWDR always performs better than the JPEG-2000 in term of color SNR. The performance

Table 3.8: Color SNR (dB) and component PSNR (dB) versus bitrate for JPEG-2000 and CWDR on *Lena*.

Bitrate (bpp)		Color SNR	Y PSNR	U PSNR	V PSNR
0.25	JPEG2K	11.43	32.83	33.43	31.13
	CWDR	11.89	32.49	34.54	31.39
0.5	JPEG2K	13.14	34.81	35.12	32.86
	CWDR	13.49	35.69	35.34	33.15
1.0	JPEG2K	14.64	37.54	36.39	33.87
	CWDR	15.00	38.24	36.53	34.95

Table 3.9: Color SNR (dB) and component PSNR (dB) versus bit rate for JPEG-2000 and CWDR on *Peppers*.

Bitrate (bpp)		Color SNR	Y PSNR	U PSNR	V PSNR
0.25	JPEG2K	14.97	32.96	33.56	33.26
	CWDR	15.23	32.48	33.28	33.52
0.5	JPEG2K	18.45	35.61	37.79	35.69
	CWDR	18.96	36.14	37.10	36.56
1.0	JPEG2K	21.70	38.52	40.80	38.07
	CWDR	22.41	39.55	40.79	39.29

gap increases with the increase of bitrate. The author's opinion is that when the bitrate becomes higher, more coefficients become significant and the context model is more efficient for U and V.

Fig 3.7 plots the decoded *Peppers* image at 0.25 bpp, *i.e.*, a compression ratio of 1/96. The difference in subjective quality of the two images cannot be distinguished.

3.5 Summary

In this chapter, we started from the performance improvement of an existing embedded wavelet image coding scheme, namely, the WDR, by decreasing its complexity as well as improving its compression ratio. We then proposed to applying a very efficient context modeling method to further increase the compression ratio. The resultant

(a) JPEG-2000 at 0.25 bpp (b) CWDR at 0.25 bpp

Figure 3.7: The decoding output of the *Peppers* at compression ratio of 96:1.

scheme achieves similar performance to the best benchmark schemes without incurring significant computation overhead. We also demonstrated that the same context modeling approach can be used on multi-component images, such as color images, so that the inter-component dependency could be exploited to increase compression gain.

Chapter 4

Parallel Image Compression Through Bit-Plane Coding

Wavelet transform based image and video coding schemes have already found wide usage in the scientific and industrial community. In many application scenarios, such as image database indexing/retrieval and image sequence compression, parallel processing capability is desirable because of the huge volume of data and the real-time processing requirement. In this chapter, we will demonstrate how we can fully exploit the wavelet pyramid data structure and realize parallel processing on all three dimensions of a wavelet pyramid, namely, pyramid width, pyramid height, and pyramid bit-depth. More importantly, the parallelization strategy proposed here is a general framework, which could *technically* be applied to *any* embedded wavelet image coder. In §4.1, we briefly introduce the status of the parallel wavelet image coding. We describe the proposed parallel encoding and sequential decoding (PESD) framework in §4.2. In §4.3, the PESD framework is extended to cover bitstream compatibility. The decoding procedure is discussed in §4.4. Experimental results are presented in §4.5.

4.1 Overview of Parallel Wavelet Image Coding

Although parallel wavelet image compression can establish itself as a sound research topic, so far a significant amount of work has been done on parallel wavelet transform algorithms, which function as the first stage of a complete image coder. When it comes to parallel processing of the quantization and the entropy coding stages, fewer studies have been reported in literature, which in the author's opinion results from the difficulty in parallelizing the zerotree-based coding methods [105, 108] since many wavelet image coding methods proposed up-to-date are based on the zerotree data structure.

Parallelization of the EZW algorithm was first reported in [20], where two approaches are proposed: (1) a straightforward parallelization which ensures that each processing element (PE) processes only entire zerotrees, and (2) one PE is reserved for the collection of the symbols that have to be encoded. The reserved PE reorders the symbols and encodes them. This approach results in higher algorithmic complexity. In fact, the first approach can be generalized as *local processing* principle under which an image (a decomposed wavelet pyramid) can be partitioned into pixel (coefficient) blocks and each block can be compressed locally by a PE. This one-block-per-PE approach is also suggested for consideration of parallel execution in the JPEG-2000 image coding system [118].

Parallelization of the SPIHT algorithm is more difficult because of the list structures that it uses. The list structures in the SPIHT cause a varying and data-dependent memory requirement. The task of memory management of adding, dropping, and removing list nodes is complicated and undesirable in memory-constraint environments. To make it worse, managing a distributed list across multiple PEs is even more difficult. Some techniques have been proposed to remove the list structures from zerotree-based coders to make them more suitable for hardware implementation and parallel processing in [56, 61, 134].

Because of the dominant popularity of the zerotree-based techniques, virtually no attention has been dedicated to parallelization of other wavelet coding schemes, which intuitively should be easier to parallelize. To be shown in this chapter, one category of such algorithms is based on run-length coding of bit-planes [112] within a wavelet pyramid [69, 86, 121]. The common features of these methods are: (1) each coefficient within a wavelet pyramid is linearly indexed by a single pointer instead of a pair of coordinates commonly used for 2-D image data, (2) a wavelet pyramid consists of bit-planes and the bit-planes are encoded progressively, and (3) each bit-plane is partitioned into several binary sequences before the binary sequences are encoded by efficient run-length coders.

4.2 Parallelization Through Data-Independent Bit-Plane Encoding

The success of wavelet schemes in the area of still image compression has been attributed to the exploitation of the statistical dependency among wavelet coefficients. For example, zerotrees in the EZW and the SPIHT, and other context models [86, 118], all utilize such dependency. One drawback of exploiting data dependency is that the intermediate state information of the coder needs to be tracked, which causes difficulties in memory management and parallelization. In the following sections, we will review the common features in [69, 86], and discuss how to design a data-independent parallel bit-plane encoder based on the PESD framework.

4.2.1 Coefficient Reordering By Linear Indexing

Linear indexing denotes the process of mapping a 2-D pyramid to a 1-D array. For a pyramid of size $W \times H$, the length of the array will be WH. Each coefficient will be assigned a unique index in the range of $[1, WH]$. Because we scan the coefficients

in the order of the indices and want to encode the coefficients with larger magnitude earlier to achieve a better rate-distortion performance, ideally we hope to: (1) assign smaller indices to larger (in terms of magnitude) coefficients, and (2) cluster large coefficients together to increase the "skewness" of the significant-symbol distribution, thus improving the efficiency of the RLC.

Since wavelet schemes are known to have good energy compaction ability, it is reasonable to assign indices beginning from the coarsest scale and ending at the finest scale. Hence, in all the techniques [69, 86], the coefficients are scanned following that order. The major difference in indexing among these techniques lies in the scan-order within each scale and/or each subband.

Used as an example for parallelization in this chapter, the MWDR proposed in §3.2 takes a simple approach in the design of the linear indexing scheme. Within the same scale (resolution), the scanning priority of subbands is $LH \rightarrow HL \rightarrow HH$. Here, LH (HL) denotes low-pass (high-pass) filtering by row/width and high-pass (low-pass) filtering by column/height. Within an HL subband, coefficients are raster-scanned column-by-column, while coefficients in all the other three types of subbands are raster-scanned row-by-row.

4.2.2 Bit-Plane Partitioning and Run-Length Encoding

As shown in Fig. 4.1, a wavelet pyramid can be seen as a "bit cube" composed of $W \times H \times (D+1)$ bits, where D is the bit-plane depth determined by the coefficient with the largest magnitude, *i.e.* $D = \lfloor \log_2 |c_{\max}| \rfloor$. In addition to D bits for the magnitude of each coefficient, one more bit is needed to record the sign information for each coefficient. The principle of embedded coding is to encode this bit cube progressively.

Bit-plane p, B_p, is composed of the p-th bit of all coefficients $\{c_i\},\ i \in [1, \ldots, WH]$, where bit-plane 0 denotes the set of the least significant bits of all coefficients. B_p is

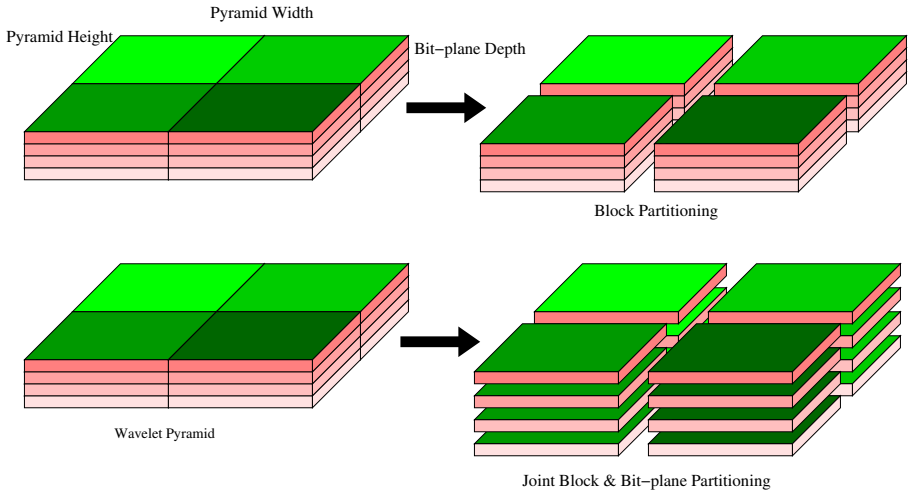


Figure 4.1: Comparison between conventional parallelization strategy and the new strategy established in this paper. The new strategy at the bottom provides another dimension of processing flexibility for an MIMD architecture.

further partitioned into two or more sets (fractional bit-planes). In the MWDR, two sets are obtained for B_p: $B_{p,s} = \{b_{p,i} | b_{q,i} = 0, \forall p < q \leqslant D\}$ and $B_{p,r} = B_p - B_{p,s}$. The set $B_{p,s}$ corresponds to coefficients that are insignificant above the bit-plane p. $B_{p,r}$ corresponds to the refinement bits for coefficients whose most significant bits have already been encoded. $B_{p,s}$ is then encoded by efficient RLC. Reduced index coding is employed as the run-length encoder in the MWDR. The sign bit of the stop symbol (significant coefficient) is inserted into the output bitstream following the encoding of each stop symbol. The sequence of bits in $B_{p,r}$ is appended behind the run-length encoding output. The bit-planes are progressively encoded until the targeted bit-rate is met.

4.2.3 Parallel Bit-Plane Encoding

Before starting the discussion on bit-plane encoding, let us appreciate the *asymmetric information availability* at the encoder and the decoder and see how it can be exploited. At the sender side, information about a complete wavelet pyramid is available to the encoder. At the receiver side, only partial information about the pyramid

is available to the decoder. To keep synchronization between the encoder and the decoder at any intermediate point, the sender side pretends that not all information about a pyramid is known and only exploits the information already sent to satisfy *causality* at the receiver.

To make it clearer, let us examine how a bit-plane of the pyramid is decoded in the MWDR. First, a sorting pass is applied to the coefficients in the insignificant coefficient set (ICS) after the encoding of the immediate higher bit-plane. Run lengths between two neighboring significant coefficients are encoded. The interesting point is that an ICS is in fact not necessary in the algorithm because the scan order is fixed and we know exactly which coefficients are excluded from current sorting pass based on their magnitudes, provided that the whole pyramid is known. If the magnitude of a coefficient is greater than $2T_p$ (T_p is the threshold for current bit-plane p), it is excluded from the sorting pass because we know it is already labeled significant in some higher bit-plane. Hence, we can follow the exact scan-order within a sorting pass without using the ICS.

It is clear now that data structures that are employed to track intermediate states of the encoders, such as zerotrees, lists, and sets, are all removable since these intermediate states can be recovered from the original wavelet pyramid by computation. This concept is *general* and holds true for *all* embedded wavelet image coders. In fact, there is always a trade-off between memory demand and computational complexity when it comes to implementing a wavelet coding algorithm.

Since intermediate states can be recovered from the original pyramid by way of computation, we no longer need to store intermediate states produced from encoding higher bit-planes to encode lower bit-planes. However, we are free from this *forced* progressiveness and are able to encode bit-planes in parallel as shown in Fig. 4.1. From this perspective, we note that this concept is general and applicable to all embedded wavelet image coders. The only question now is how much computational

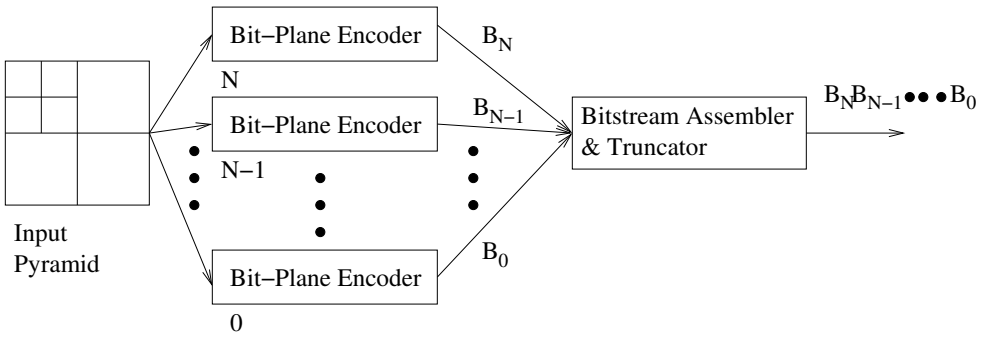


Figure 4.2: Processing architecture of the proposed bit-plane encoder, where B_n denotes the generated bitstream for the n-th bit-plane.

cost it takes to recover the intermediate state information.

The major problem for bit-plane encoding is the refinement pass. Coefficients that need refinement are stored in the significant coefficients set (SCS) in the WDR. Within the SCS, the coefficients are ordered according to their MSBs and their indices determined by the scan-order. If we want to follow the same order as in the SCS in the bit-plane encoding, we need to reorder those already-significant coefficients and construct a SCS. Here, we argue that each output bit within a bit-plane contributes on the same order to the reduction of distortion. Hence, the identical output order of the refining bits is indeed not necessary. This coding strategy causes the bitstream incompatibility between the PESD and the MWDR. However, we need to emphasize that bitstream compatibility is not difficult to achieve if we would like to sacrifice some buffer size and algorithmic complexity. We will cover the bitstream compatibility in the next section.

The output bitstream from each bit-plane encoder consists of the output of the sorting pass followed by the output of the refining pass. The output from each bit-plane encoder is buffered before they are ultimately assembled by a bitstream assembler and truncated to fit the rate budget, as shown in Fig. 4.2.

The pseudo-codes for a bit-plane encoder are shown in Algorithm 2.

The output of the run-length bits and the output of the refining bits are not in-

Algorithm 2 The bit-plane encoder.

```
/*P is a pointer to the wavelet pyramid*/
/*b is an integer indexing current bit-plane*/
bit_plane_encoder(P,b) {
    run_length := 0; T := 2^b;
    for i := 1 to length(P) do {
        (x, y) := index_to_coordinates(i);
        if(|P(x,y)| < T) run_length++;
        else if(|P(x,y)| ⩾ 2T) write_refining_bit(P(x,y), b);
        else {
            write_run_length_bits(run\_length);
            write_sign_bit(sign(P(x,y)));
            run_length := 0;
        }
    }
    write_end_marker(run_length);
}
```

terleaved in the final produced bitstream. Actually, the output of these two different types of data are buffered separately before the refining bits are finally appended to the run-length bits. Note that the listing above only describes the algorithm for a bit-plane encoder. To encode the entire wavelet pyramid, this procedure is called multiple times (concurrently or sequentially) within a main program when we talk about software implementation. The decomposed wavelet pyramid P and the bit-plane number b are passed to the procedure as the only parameters. A nice feature of this algorithm is that no communication among bit-plane encoders is needed, resulting in a very simple parallel architecture since each PE can process individual bit-plane(s) independently without resorting to information from other PEs when this algorithm is implemented with parallel processing hardware.

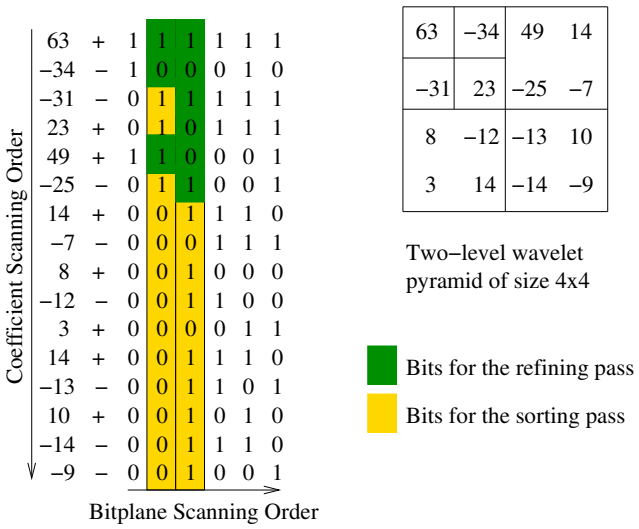


Figure 4.3: One example to illustrate the fixed scan-order and the bit-plane encoding.

4.3 Bitstream Compatibility in Parallel Bit-Plane Encoding

In last section, we have explained that an intermediate list like the ICS is not necessary since along the path of the fixed scan-order, we know exactly which coefficients should be skipped solely by testing their magnitudes. Note that this magnitude test is conducted on all coefficients in the PESD while only coefficients in the ICS undergo such test in the MWDR. However, the extra computational cost of a magnitude test is just one logic instruction, which is justified compared to the memory requirement and computational load imposed by managing dynamic list structures.

We have mentioned before that the PESD is designed for parallel encoding and the algorithm described in §4.2.3 generates a bitstream that is incompatible with the bitstream generated by the original MWDR. However, we now present a bitstream compatible algorithm with which the original MWDR decoder works just fine.

Before we give the pseudo-code listing, one example is illustrated in Fig. 4.3 to show how bitstream compatibility increases the encoder complexity.

The encoding of bit-planes B_3 and B_4 is shown in Fig. 4.3. Those bits falling into the two different passes, *i.e.*, the sorting pass and the refining pass, are labeled in different levels of shade. From the figure, we can see that the bits in the sorting pass are scanned in the same order. However, the bits in the refining pass are scanned differently compared to the MWDR. When we encode B_4, the refining bits are scanned as $\{1_{63}, 0_{-34}, 1_{49}\}$. Similarly, the refining bits are scanned as $\{1_{63}, 0_{-34}, 1_{-31}, 0_{23}, 0_{49}, 1_{-25}\}$ when B_3 is encoded. This shows that the position of the refining bits in the output bitstream is solely determined by the scan-order, which can be expressed as:

$$\begin{cases} I(c(x,y))_{p,s} &= f(S), \quad |c(x,y)| < 2T_p \\ I(c(x,y))_{p,r} &= f(S), \quad |c(x,y)| \geq 2T_p \end{cases} \tag{4.1}$$

where $I(c(x,y))_{p,s}$ denotes the index of $c(x,y)$ in the sorting pass of B_p if the magnitude of $c(x,y)$ is less than two times of the threshold T_p for B_p. Similarly, $I(c(x,y))_{p,r}$ denotes the index of $c(x,y)$ in the refining pass of B_p. The $f(\cdot)$ denotes an index mapping function while S represents the scan-order.

When we use the MWDR to encode the pyramid in Fig. 4.3, the scan order for the sorting pass stays the same. However, the order for the refining bits is changed because the significant coefficients in B_4 are stored in the SCS and they get refined first when we output refining bits for B_3. Now the refining bits are scanned as $\{1_{63}, 0_{-34}, 0_{49}, 1_{-31}, 0_{23}, 1_{-25}\}$. It is clear that positions of the refining bits are not only determined by the fixed scan-order but also determined by the refining bits ordering of the higher bit-planes.

$$\begin{cases} I(c(x,y))_{p,s} &= f(S), & |c(x,y)| < 2T_p \\ I(c(x,y))_{p,r} &= f(S, \mathbf{B}_{\{q|q>p\}}), & |c(x,y)| \geq 2T_p \end{cases} \tag{4.2}$$

To make the bitstream compatible with that of the MWDR, we need to output the refining bits according to the order set by the SCS. Since no list is available in

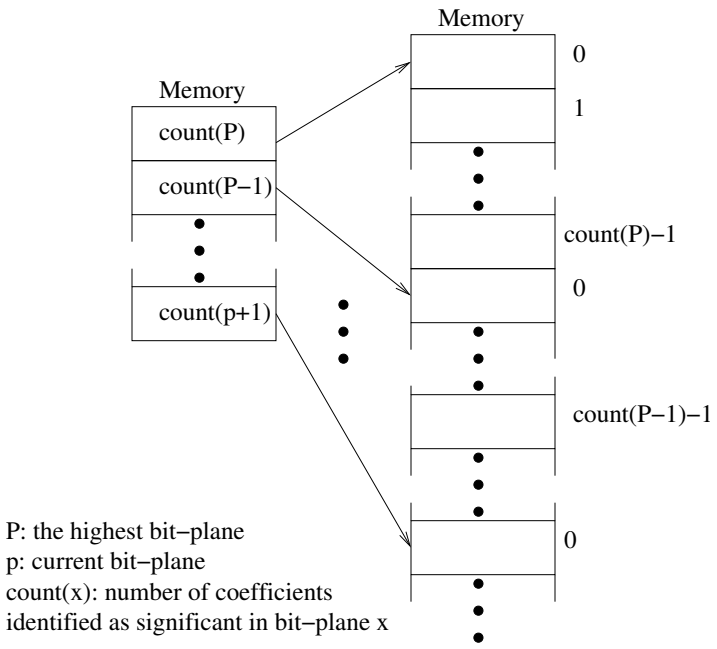


Figure 4.4: The data structure that stores the order of the refining bits for the encoding of B_p.

the PESD, we need to parse all the coefficients to collect information that is provided by the SCS in the MWDR. This parsing stage now makes our algorithm a two-pass algorithm. In the first pass, we collect information on how to order the refining bits. Then in the second pass, we output refining bits according to the order set after the first pass. Fig. 4.4 illustrates the data structure that handles the output order of the refining bits.

In the first pass, we need to determine the priority of the refining bits, *i.e.*, those coefficients which are significant in the highest bit-plane B_P will get refined first, following are the coefficients significant in B_{P-1}. Significant coefficients from bit-plane B_{p+1} will be refined last.

When the first pass is completed, the total number of refining bits for B_p and the numbers of significant coefficients for all higher bit-planes are known. Hence, we can allocate dynamic memory for all refining bits of B_p and the offset for each bit-plane can be calculated based on the significant coefficients in each bit-plane. When we

output a refining bit $b_{p,i}$, its position in the bitstream is jointly determined by the highest bit-plane $p_{\max(c_i)}$ reached by the corresponding coefficient c_i, the offset for refining bits in $p_{\max(c_i)}$ in the buffer, and the index i in the fixed scan-order.

The pseudo-codes for a bitstream compatible bit-plane encoder is shown in Algorithm 3.

Algorithm 3 The bitstream-compatible bit-plane encoder.

```
/*P is a pointer to the wavelet pyramid*/
/*b is an integer indexing current bit-plane*/
/*b_max is the integer indexing the highest bit-plane*/
bit_plane_encoder(P,b,b_max) {
    run_length := 0; T := 2^b;
    for i := 1 to length(P) do { /*first pass*/
        (x,y) := index_to_coordinates(i);
        if(|P(x,y)| ⩾ 2T) {
            total_refining_bits++;
            plane := floor(log2 |P(x,y)|);
            refining_bits_count[plane]++;
        }
    }
    pointer := allocate_memory(total_refining_bits);
    for i := (b+1) to b_max do {
        offset[i]:=calc_offset(pointer,refining_bits_count);
    }
    for i := 1 to length(P) do { /*second pass*/
        (x,y) := index_to_coordinates(i);
        if(|P(x,y)| < T) run_length++;
        else if(|P(x,y)| ⩾ 2T)
            write_refining_bit(P(x,y),b,pointer,offset);
        else {
            write_run_length_bits(run_length);
            write_sign_bit(sign(P(x,y)));
            run_length:=0;
        }
    }
    write_end_marker(run_length);
}
```

4.4 Sequential Bit-Plane Decoding

Contrary to the encoding process that can be truly parallelized, the decoding process is inherently sequential because the information is incrementally available to the receiver following the progressive decoding of bit-planes. However, by proper algorithm analysis, we can still potentially optimize the execution of the decoding process.

The pseudo-code for a bit-plane decoder is listed in Algorithm 4. Note that it is assumed that all bit-planes higher than b have already been decoded error-free. When we decode each bit-plane, the bitstream generated by the sorting pass is first decoded and the decoded run-length and sign information is stored into a run-length list and a sign list. The wavelet pyramid is then scanned and the refining bits are read to refine the coefficients at the correct positions. When the incremented run-length equals one value in the run-length list, the coefficient in current position becomes significant and its value is initialized by current threshold and the corresponding sign. Note that the decoder is still designed for multiple bit-plane decoders, which can be executed on a parallel system. However, communication among the PEs is now inevitable since there is dependency among the bit-plane decoding processes.

One possible parallelization strategy is to label the size and the offset of the substreams for each bit-plane within the final bitstream when we encode an image and segment them later to feed them into multiple PEs. The section of the codes in a bit-plane decoder that do not depend on the results from higher bit-planes can be executed in parallel. In the remaining code section when we decode B_p, we need to know how many coefficients fall into the sorting pass and the refining pass, which depends on the correct decoding of bit-planes higher than B_p. Another strategy can be pipelining. From Algorithm 4, it can be seen that the source codes of a bit-plane decoder can be roughly treated as consisting of two major sections. The first section is the generation of an intermediate list which contains the decoded run-length and

Algorithm 4 The pseudo-code for the bit-plane decoder.

```
bit_plane_decoder(P,b) {
    while end_marker_reached() == False do {
        read_in_run_length_to_list();
        read_in_sign_bit_to_list();
    }
    run_length := 0; T := 2^b;
    significant_length := first_run_length_in_list();
    for i := 1 to length(P) do {
        (x,y) := index_to_coordinates(i);
        if(|P(x,y)| ⩾ 2T) read_one_refining_bit();
        else run_length++;
        if(run_length == significant_length) {
            P(x,y) := 1.5 · T·(next_sign_bit_in_list() ? −1:1);
            significant_length := next_run_length_in_list|();
            run_length := 0;
        }
    }
}
```

sign information, which does not depend on the results from higher bit-planes. The second section is the reconstruction process that does depend on the decoding of higher bit-planes. These two sections of codes can be efficiently pipelined.

To decode the bitstream generated by the bitstream compatible bit-plane encoder, the MWDR decoder can be directly employed without any modification. Alternatively, a two-pass bit-plane decoder similar to that described in Algorithm 4 can be designed. In the first pass of such a decoder, a list like the SCS is constructed before any refining bit is read.

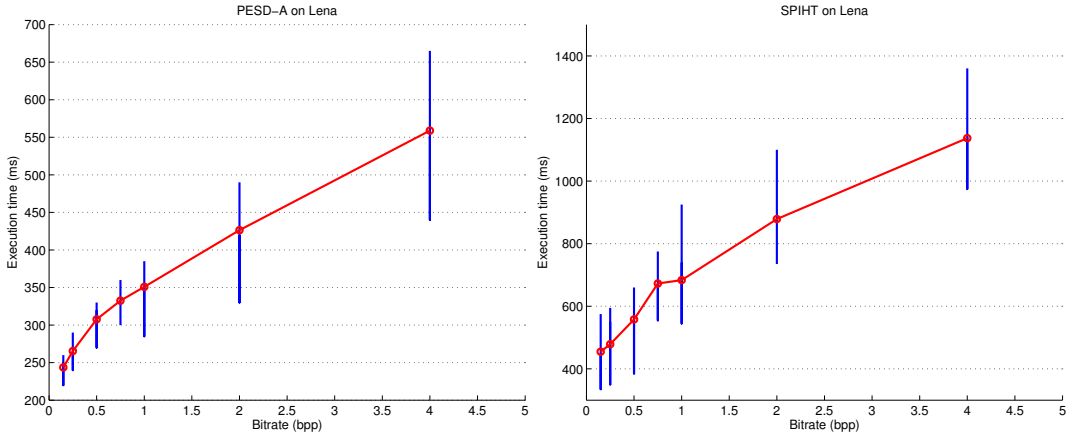
4.5 Experimental Results

To verify the correctness of the PESD, two versions of the algorithm were implemented. The first, PESD-A, is a non-threaded version. The second, PESD-B, is a threaded version which uses the Pthread library for the Linux operating system and is able to take advantage of the symmetric multi-processor (SMP) kernel of Linux.

To demonstrate the simplicity of the proposed algorithm, the execution time of the non-threaded PESD-A was compared with that of the QccPack [28] implementation of SPIHT without AC, which is well-known due to its speed and PSNR performance. This implementation [28] of SPIHT without AC was chosen because there is no AC stage in the PESD and our source codes are also based on the QccPack library. This arrangement can roughly ensure a fair comparison between the PESD and the SPIHT. We encoded two gray-scale images, *Lena* and *Barbara*, at various bitrates and used the profiling tool *gprof* [34] to record the execution time. The size of both images is 512×512 at 8 bpp. Each image was consecutively encoded thirty times at each bit-rate. The program was run on a Pentium III 800-MHz workstation with 256 MB PC-100 SDRAM running Linux uni-processor (UP) kernel 2.4.20-8 and GCC 3.3.2-1.

Fig. 4.5 illustrates the mean value and the range of samples at each bitrate for the two algorithms. Fig. 4.6 compares the mean execution times of the two methods at each bitrate on same scale. The mean execution time are compared in Fig. 4.7 for the coding of *Barbara*. The reconstructed images processed by the PESD-A at very low bitrate can be visually compared with those processed by the SPIHT in Fig. 4.8.

To demonstrate the potential of the parallel bit-plane encoding, the PESD-B implements the bit-plane encoders as multiple threads. We ran the PESD-B on a Pentium III 700-MHz dual-processor server with 256 MB PC-100 SDRAM running Linux SMP kernel 2.2.24-7.0.3 and GCC 2.96. Since the PESD-B is a threaded implementation, we expected that bit-plane coding can be accelerated since threads can be scheduled to run on both processors at the same time. To demonstrate how threading improves timing performance, we ran the PESD-A, the PESD-B, and the PESD-B with bitstream compatibility (PESD-B/BC) on the server. It is worth noting that we did not try to optimize any of the three implementations. All bit-planes are encoded in the PESD-B and the PESD-B/BC before the bitstreams



(a) (b)

Figure 4.5: The mean and the range of samples of the execution time of the SPIHT and the PESD-A on *Lena*.

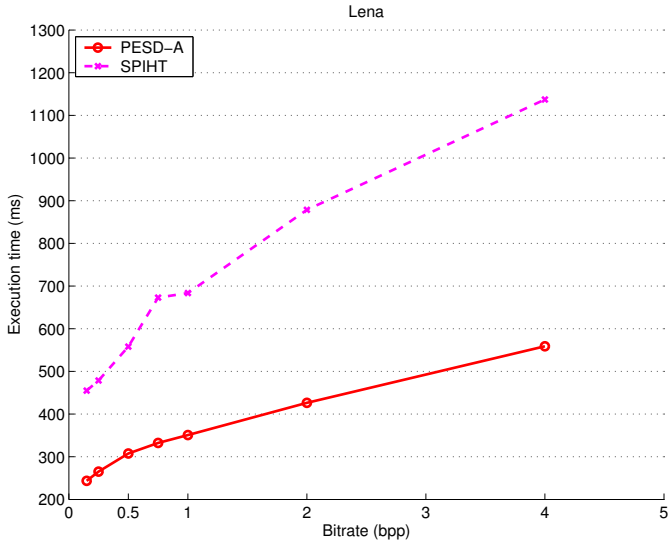


Figure 4.6: Comparison of execution time of the SPIHT without AC and the PESD-A on *Lena*.

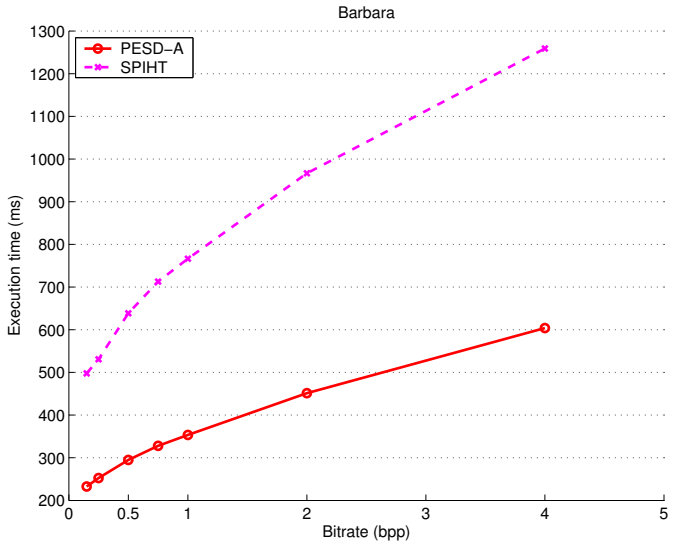


Figure 4.7: Comparison of execution time of the SPIHT without AC and the PESD-A on *Barbara*.

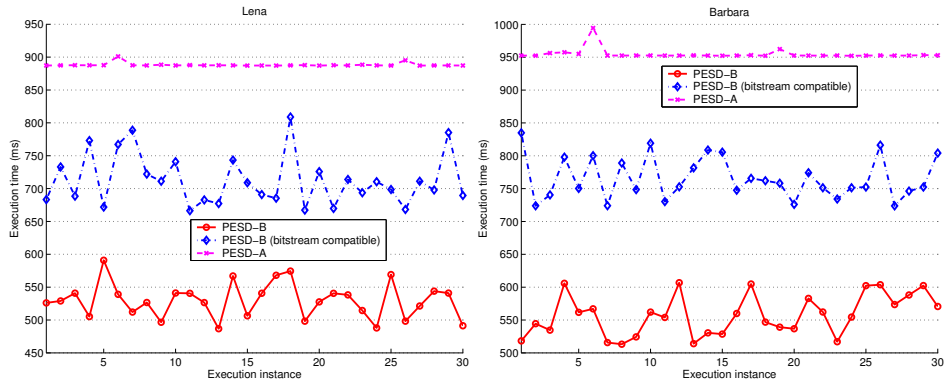
are assembled and truncated to fit rate budget. However, since the PESD-A is non-threaded, the bit-plane encoding in the PESD-A is actually sequential and the bit-plane encoding can be stopped once the rate budget is met. To make the results more revealing, we encoded all bit-planes using the PESD-A to compare the timing performance. Since threaded program profiling is not supported by the gprof tool, we used an alternative method to collect timing information. In the PESD-A, a timer is reset just before the start of bit-plane encoding and stopped once all bit-planes are encoded. In the PESD-B, a timer is reset just before the main thread creates all children threads and stopped once all children threads are joined. The timing result of running both schemes thirty times is shown in Fig. 4.9, where the PESD-B almost doubled the speed on bit-plane encoding because of the presence of two processors.

For the PSNR quality of the PESD, it can be found in Table 4.1 that for tested images at any bitrate, the PSNR gap between the PESD and the MWDR is smaller than 0.001 dB while the gap is exactly 0 dB if an integer number of bit-planes are encoded. Note that the PSNR performance of the PESD-B/BC is identical to that of the MWDR. As shown in Fig. 4.9, to keep the bitstream compatible with MWDR,

(a) (b)

(c) (d)

Figure 4.8: Comparison on visual quality of reconstructed *Lena* and *Barbara* when they are encoded by the PESD-A and the SPIHT at 0.15 bpp.



(a) (b)

Figure 4.9: Comparison of execution time on a dual-processor server using the PESD-A, the PESD-B, and the PESD-B with bitstream compatibility using *Lena* and *Barbara*.

an additional 30% to 40% percent of processing time is required while encoding all bit-planes. This additional processing time obviously does not bring back any PSNR improvement as shown in Table 4.1. This proves our hypothesis that the order of the refinement bits is not a factor that can significantly influence the final reconstruction quality.

4.6 Summary

In this chapter, we develop a wavelet image coding algorithm that is based on the parallel execution of multiple bit-plane encoders. In addition to the conventional parallelization strategy of segmenting a wavelet pyramid into multiple code-blocks, we show that using our proposed algorithm, each code-block can be further segmented into multiple bit-planes which can be encoded simultaneously. The output bitstreams of the bit-plane encoders are buffered, assembled, and finally truncated to meet the rate budget. The proposed algorithm has been implemented as a threaded Linux application program and successfully tested on a dual-processor workstation. We also

Table 4.1: Comparison of PSNR values for the SPIHT, the SPIHT-AC [28], the MWDR, and the PESD.

Image		Output bit rate (bpp)			
		0.15	0.25	0.5	1.0
Lena	SPIHT	31.33	33.62	36.74	39.90
	SPIHT-AC	31.71	34.00	37.11	40.29
	MWDR	31.1257	33.2419	36.3603	39.5416
	PESD	31.1248	33.2419	36.3603	39.5416
Goldhill	SPIHT	28.59	30.16	32.57	35.86
	SPIHT-AC	28.89	30.44	32.96	36.38
	MWDR	28.6421	30.2451	32.6099	35.8604
	PESD	28.6419	30.2451	32.6094	35.8604
Boat	SPIHT	28.37	30.34	33.74	38.34
	SPIHT-AC	28.63	30.74	34.20	38.91
	MWDR	28.3197	30.3450	33.5517	38.2289
	PESD	28.3198	30.3440	33.5517	38.2285
Barbara	SPIHT	25.50	27.60	31.60	36.79
	SPIHT-AC	26.08	28.20	32.20	37.42
	MWDR	25.6714	27.2310	31.2009	36.0531
	PESD	25.6714	27.2307	31.2009	36.0530

show by results that in embedded wavelet image coding the ordering of refinement bits does not significantly influence the PSNR result of a reconstructed image. To the best of our knowledge, this algorithm is the first to realize parallelization through the simultaneous encoding of multiple bit-planes.

Chapter 5

Wavelet Scalable Video Coding

Recently, research interest in the scalability of compressed video has been sparked by diverse application scenarios. Incorporating scalability into compressed video makes it possible to "encode once and decode everywhere" so that an encoder does not need to know in advance that under which circumstances the compressed video will be decoded. These scalable schemes have applications in digital libraries, video streaming over networks, video telephony and conferencing, and broadcasting of regular resolution TV and HDTV. In §5.1, a wavelet spatial-scalable video coder, which fully exploits the multi-resolution characteristic of wavelet, is proposed. This algorithm employs ME/MC in the wavelet domain and realizes a layer of resolution at each wavelet decomposition level. Since there are multiple levels of ME/MC producing motion information, backward motion compensation is employed to eliminate the need for the compression and transmission of motion information. With the ever-increasing popularity of Internet video streaming, the concept of fine granular scalability (FGS) [59] has been introduced to handle the bandwidth fluctuation caused by a large number of users sharing a common physical communication channel. We propose a novel wavelet fine granular scalable (WavFiGS) video coder in §5.2. The WavFiGS consists of a base layer (BL) coder that is based on macro-block stack-run

(MBSR) coding and an error-resilient enhancement layer (EL) coder that is based on modified Golomb coding [33, 99].

5.1 Spatial Scalable Video Coding with Hierarchical Backward Motion Compensation

In conventional hybrid video coders, spatial scalability is usually achieved by down-sampling of the original frames [41], either in the spatial domain or the transform domain. This down-sampling process is a lossy process and non-invertible. However, a wavelet spatial-scalable video coder can readily achieve several layers of spatial-scalability readily by combining the multi-resolution capability of wavelets and the so-called *In-Band* ME/MC techniques.

5.1.1 Backward Motion Compensation

The word *Backward* here may cause confusion, and thus requires some clarification. Many people may perceive this word as defined in the video coding standards, *e.g.*, MPEG-4 [43]. In those standards, backward motion compensation (BMC) means that to predict a B-frame from a P-frame, the latter is decoded before the B-frame, yet displayed after it. In our proposed scheme, backward motion compensation means one frame was predicted from the previous decoded two frames, both of the previous frames reside temporally ahead of the current frame in the raw sequence order.

Fig. 5.1 illustrates a two-level wavelet decomposition of an image. In order to perform hierarchical backward ME, the low-pass subband at the coarsest resolution of the wavelet pyramid is obtained first. Taking Fig. 5.1 as an example, we assume that LL_2 is already known to the decoder before the ME. To demonstrate the procedure of ME/MC, we use k as a frame index. Let $LL_{k-1,2}$ denote the second-level low-pass subband of the reference frame and $LL_{k,2}$ denote the low-pass subband of the current

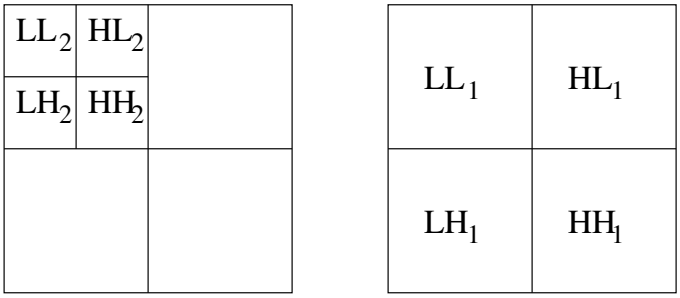


Figure 5.1: The two-level wavelet decomposition and the subband naming convention.

frame at the same level. We use the following equation to denote the ME between two subbands:

$$MV_{k,n} = ME(LL_{k-1,n}, LL_{k,n}) \tag{5.1}$$

where MV denotes motion vectors.

Because of the aliasing effect [12], we cannot directly apply $MV_{k,2}$ to the high-pass subbands at Level 2 of the pyramid of Frame $k-1$ to obtain the predicted high subbands in the pyramid of Frame k at the same level, which are denoted as $\overline{HL}_{k,2}$, $\overline{LH}_{k,2}$, and $\overline{HH}_{k,2}$. Instead, both $LL_{k-1,2}$ and $LL_{k,2}$ are up-sampled and filtered to generate two new subbands, $LL^{\uparrow}_{k-1,2}$ and $LL^{\uparrow}_{k,2}$, which have the same size with LL_1 and serve as approximations of $LL_{k-1,1}$ and $LL_{k,1}$. ME is conducted between $LL^{\uparrow}_{k-1,2}$ and $LL^{\uparrow}_{k,2}$. The resultant motion vectors are applied to $LL_{k-1,1}$ to obtain a predicted subband $\overline{LL}_{k,1}$. Then, $\overline{LL}_{k,1}$ is decomposed by one level to generate four subbands, and the three high-pass subbands serve as the prediction for $HL_{k,2}$, $LH_{k,2}$ and $HH_{k,2}$. The prediction error, *i.e.*, displaced subband difference (DSD), is then quantized and entropy coded. Fig. 5.2 illustrates the ME scheme proposed in [141]. For simplicity, only one-dimensional processing is shown while the input video signal is 2-D.

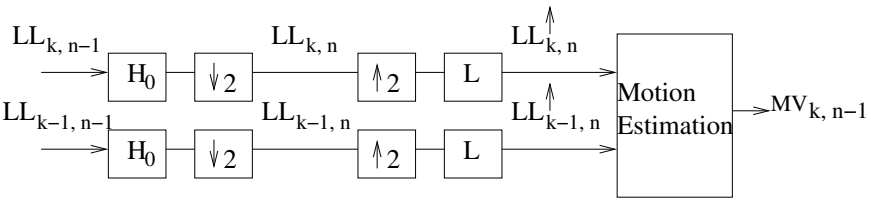


Figure 5.2: Block diagram for one-level of ME. H_0 is the low-pass analysis filter. L is the low-pass interpolation filter. Note that for simplicity, only one-dimensional filtering, decimation, and interpolation are shown.

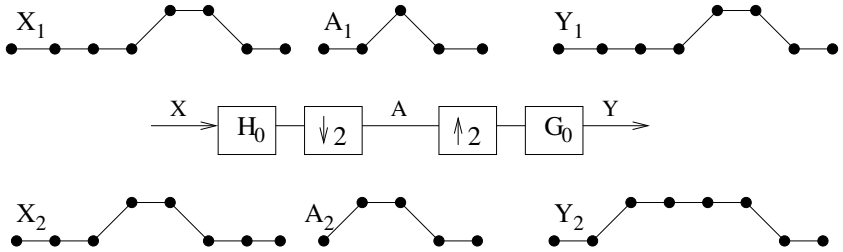


Figure 5.3: Examples of the coefficients from the *Haar* wavelet for a 1-D signal $X_1[n]$ and its shifted-by-one-grid equivalent $X_2[n]$. H_0 is the low-pass analysis filter and G_0 is the low-pass synthesis filter. The amplitudes of the coefficients are not exactly as shown. Here, we are more interested in the shape of the output signal.

5.1.2 New Low-Band-Shifting-Based Scheme

Although purely backward ME/MC scheme is achieved in [83, 141], Fig. 5.3 shows that the scheme illustrated in Fig. 5.2 cannot estimate the shifting of a signal. Here, the usage of a block-matching algorithm (BMA) is assumed. From Fig. 5.3, we can see that the aforementioned scheme can only estimate the shifting of a signal by even pixels. Another observation is that although the whole reconstructed reference frame is available to the decoder, the aforementioned scheme only uses the LL subband at coarser level to generate an up-scaled subband for ME at the next finer level. There may exist a better way to utilize the information available to the decoder.

It is fairly straightforward to notice in Fig. 5.3 that we can still predict $X_2[n]$ if we shift $X_1[n]$ by one grid before conducting ME. This is the basic concept behind the so-called *LBS method*, which is proposed by Park and Kim [89] to deal with the shift-

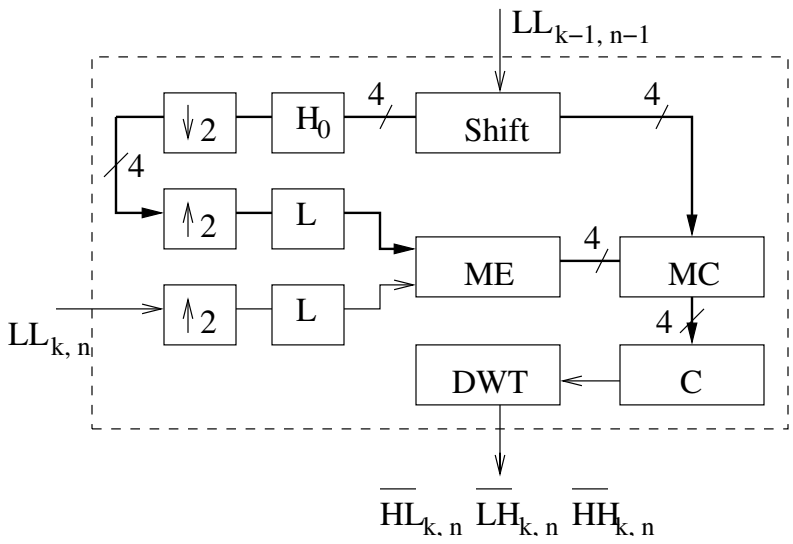


Figure 5.4: The building block of the proposed ME/MC scheme. C denotes combination of the four predicted subbands.

variant property of the wavelet decomposition. However, the authors constructed a new data structure, called a "wavelet block", to conduct forward ME, which by itself is not hierarchical.

It is observed in Fig. 5.3 that the reference signal $X_1[n]$ and the approximation of the current signal $X_2[n]$, namely, $A_2[n]$ are available to the decoder before ME. What we need is to shift $X_1[n]$ by one grid to generate $X_1^s[n]$, which will be filtered and decimated to generate $A_1^s[n]$. Both $A_1[n]$ and $A_1^s[n]$ are used in ME/MC to generate two predictions of $A_2[n]$. These two predictions are then combined to generate a final prediction. When we extend this method to a 2-D wavelet subband, we can shift the subband horizontally, vertically, and diagonally by one grid to obtain three shifted subbands. We denote these three shifted subbands as LL^h, LL^v, and LL^d. The original subband is denoted as LL^a. Now, instead of having only one reference LL subband, we have four reference LL subbands available at each level for MC. Fig. 5.4 illustrates the ME/MC block at each level.

The design of the *combination* block in Fig. 5.4 remains an open problem. In our simulations, we simply took the average of the four predicted subbands. The overall

proposed algorithm is described as follows:

1. Decompose the current frame I_k into an N-level wavelet pyramid.

2. Quantize the subband difference (SD) between $LL_{k-1,N}$ and $LL_{k,N}$ to form the reconstructed low-pass subband $\widetilde{LL}_{k,N}$. Set $n = N$.

3. Apply the ME/MC algorithm described in Fig. 5.4 to generate three predicted high-pass subbands: $\overline{HL}_{k,n}$, $\overline{LH}_{k,n}$, and $\overline{HH}_{k,n}$.

4. Quantize the displaced subband difference (DSD) of the three high-pass subbands and form the reconstructed three high-pass subbands.

5. Perform wavelet synthesis using $\widetilde{LL}_{k,n}$, $\widetilde{HL}_{k,n}$, $\widetilde{LH}_{k,n}$, and $\widetilde{HH}_{k,n}$ to obtain $\widetilde{LL}_{k,n-1}$.

6. Decrement n by 1. If $n \neq 0$, go back to Step 3.

For wavelet-based still image coding schemes, the zerotree-based structure [108, 105] that exploits cross-scale statistical dependency is very efficient and successful. But direct application of the zerotree data structure in our scheme is very difficult, if not impossible. There are two problems that are hard to solve. First, the ME/MC are conducted at different levels of a wavelet pyramid. This means that the residual subbands at level n are not obtained from filtering and decimating the residual LL_{n-1} subband. So there is not much correlation that remains to be exploited among subbands at different levels. Second, we must ensure the synchronization between the encoder and the decoder; no information is allowed to be used by the encoder if it is not available to the decoder. At each level, both the encoder and the decoder demand availability of a reconstructed copy of current frame at the lower resolution (higher level). This requirement implies that we cannot obtain the whole residual pyramid to construct the zerotree hierarchy before quantization. Unlike usual transform-based hybrid video coding scheme, in which we conduct ME/MC before quantizing

the whole frame, here we have a sequence of interleaved operations, *i.e.*, ME/MC $\rightarrow$ quantization $\rightarrow$ reconstruction $\rightarrow$ ME/MC $\rightarrow$ quantization $\rightarrow$ reconstruction $\rightarrow \cdots$. To solve this problem, Nosratinia and Orchard utilized a predetermined zerotree in the encoder, which is an estimation of the zerotree of the residual pyramid [83] and there is no way to ensure the accuracy of this estimation. Thus, an ideal quantization scheme for the proposed coding framework must be efficient even if we need to encode subbands independently.

We implemented an adaptive context modeling estimation-quantization (EQ) wavelet image coder proposed in [64] to encode each subband in the residual pyramid. In this adaptive EQ coder, wavelet coefficients in each subband are modelled as drawn from an independent generalized Gaussian distribution (GGD) field. Each subband fits into a GGD field of fixed-shape, zero-mean, and locally slow-varying variances. The probability density function (pdf) of zero-mean GGD is [49]:

$$f(x) = \frac{\nu}{2\sigma\Gamma(\nu^{-1})}\sqrt{\frac{\Gamma(3\nu^{-1})}{\Gamma(\nu^{-1})}} \cdot \exp\left(-\left(\sqrt{\frac{\Gamma(3\nu^{-1})}{\Gamma(\nu^{-1})}}\left(\frac{\mid x \mid}{\sigma}\right)\right)^{\nu}\right) \quad (5.2)$$

where σ denotes the standard deviation and ν is the shape parameter. For the quantization of a coefficient in the subband, it is assumed that the coefficient and its causal neighbors share an identical variance. Thus, a maximum-likelihood estimate of the variance of the coefficient is computed based on a causal and quantized neighborhood context. This ensures that an encoder and a decoder keep synchronization because the decoded causal neighbors are available to decoder before the decoding of current coefficient. Once the estimated variance is obtained, it is used to index the entries of a look-up table. Each entry of the table corresponds to a specific quantizer, which is designed off-line according to rate-distortion criterion.

Before proceeding to the design of an optimal quantizer, it is necessary to determine which GGD model to use for a given subband. In our scheme, the only

parameter we need to specify for the whole subband is the shape parameter ν. We use the kurtosis K of a zero-mean GGD for the purpose of shape-fitting [51]:

$$K = \frac{\langle x^4 \rangle}{(\sigma^2)^2} = \frac{\Gamma(5\nu^{-1})\Gamma(\nu^{-1})}{\Gamma(3\nu^{-1})^2}. \tag{5.3}$$

An important feature of the EQ coder is that the optimal rate-distortion (R-D) quantizers are designed in advance as a function of shape parameters ν, zero-mean ($m = 0$) and unit variance ($\sigma^2 = 1$) and stored as a look-up table. For small ν, the pdf has a sharp peak. The Laplacian density corresponds to $\nu = 1$, while the Gaussian density corresponds to $\nu = 2$. We used the same shape parameter set as in [49], $\nu \in \{0.5\ 0.6\ 0.7\ 0.8\ 0.9\ 1.0\ 1.5\ 2.0\}$. So, we need to design a set of normalized optimal quantizers for the conditional probability density function $p(x|m, \sigma, \nu)$, where $m = 0$ and $\sigma = 1$.

We choose to design normalized dead-zone scalar quantizers that are characterized by step size s and dead-zone ratio α. Consider an input random variable X having a pdf of $p(x|0, 1, \nu)$, let an n-level dead-zone quantizer $Q^{(n)}(\cdot)$ be defined in terms of its n output values and quantization indices:

$$y_i^{(n)} = \begin{cases} (-\frac{\alpha}{2} + \beta + i - i_0) \cdot s, & 0 \le i < i_0 \\ 0, & i = i_0 \\ (\frac{\alpha}{2} - \beta + i - i_0) \cdot s, & i_0 < i \le n-1 \end{cases} \tag{5.4}$$

where i_0 denotes the quantization index of the zero bin. In our case, $i_0 = \frac{n-1}{2}$. Because a large number of coefficients having values around zero demands an odd value of n. The parameter β is a real number in the interval [0, 1], which tunes the output value of quantization bins. The parameter β has no impact on the final bitrate, but influences the reconstruction distortion. The quantization index i is

determined by

$$i = \begin{cases} i_0 - \lceil -\frac{x}{s} - \frac{\alpha}{2} \rceil, & x < -\frac{\alpha s}{2} \\ i_0, & -\frac{\alpha s}{2} \leq x \leq \frac{\alpha s}{2} \\ i_0 + \lceil \frac{x}{s} - \frac{\alpha}{2} \rceil, & x > \frac{\alpha s}{2} \end{cases} \tag{5.5}$$

while the notation $\lceil \cdot \rceil$ denotes a ceiling operation.

The well-known Lagrangian cost function was employed in the design of the optimal quantizers

$$J^{(n)} = D^{(n)} + \lambda R^{(n)}, \tag{5.6}$$

where the parameters $J^{(n)}$, $D^{(n)}$, and $R^{(n)}$ are defined as follows. The expected quantizer distortion $D^{(n)}$ is defined by

$$D^{(n)} = \sum_{i=0}^{n-1} \int_{t_{i,low}^{(n)}}^{t_{i,high}^{(n)}} (x - y_i^{(n)})^2 f(x) dx \tag{5.7}$$

$t_{i,low}^{(n)}$ and $t_{i,high}^{(n)}$ are the two thresholds that determines the i-th bin. In our dead-zone quantizer, $t_{i,high}^{(n)} - t_{i,low}^{(n)} = s$ when $i \neq i_0$. Based on $t_{i_0,high}^{(n)} = \frac{\alpha s}{2}$, $t_{i_0,low}^{(n)} = -\frac{\alpha s}{2}$, and $t_{i+1,low}^{(n)} = t_{i,high}^{(n)}$, we can recursively determine the thresholds for all bins. The rate is defined by the output entropy of the quantizer

$$R^{(n)} = -\sum_{i=0}^{n-1} p_i^{(n)} \log_2 p_i^{(n)} \tag{5.8}$$

where $p_i^{(n)}$ is the probability of reconstruction value $y_i^{(n)}$,

$$p_i^{(n)} = \int_{t_{i,low}^{(n)}}^{t_{i,high}^{(n)}} f(x) dx \tag{5.9}$$

From equations above, it is clear that once λ is fixed, we can always find an optimal solution in terms of a minimized J by adjusting s, α, and β for an n-level dead-zone scalar quantizer. Since we can obtain an optimal quantizer for any λ, we are

able to generate a look-up table that corresponds to a large number of discrete λ values. Note that these quantizers are all normalized to the unit variance. In order to quantize a real subband coefficient c with a target λ, we need to use the scaled value $\lambda^\star = \lambda/\hat{\sigma}^2$ as index to the table. The parameter $\hat{\sigma}^2$ is the variance estimate of c based on quantized causal neighbors. The step-size $s_{\lambda^\star}$ of the indexed optimal quantizer needs to be scaled back to $\hat{\sigma}s_{\lambda^\star}$ to quantize c [64]. The symbol probabilities associated with the quantization indices for different GGDs are also pre-stored in a separate table and indexed to be used in arithmetic encoding [135] of the quantization indices.

The performance evaluation of the proposed technique is now presented and compared with two other benchmark schemes. The first benchmark scheme is the scheme proposed in [141], in which we used the interpolation filter of length 7 as proposed by the authors. The second scheme is the widely used MPEG-2 codec software [80]. In the experiments we used the luminance component of the full-motion *Football* SIF sequence of size 352×240 at 30 fps. The bi-orthogonal 9/7-tap filter-bank [2] was used in the decomposition. The decomposition level was set to $N = 3$, which means that the size of the highest-level subbands is 44×30. The temporal distance was set to 1, *i.e.*, the ME/MC was done between the decomposed reconstructed previous frame $\widetilde{I}_k$ and decomposed current frame I_{k+1}. Also, in order to demonstrate the efficacy of the proposed technique as well as keeping the source coding work simple, we did not use B frames in the prototype coder. To maintain a fair comparison among the aforementioned three schemes, B frames were also precluded from the MPEG-2 codec. The number of frames in one group of pictures (GoP) was set to 30, *i.e.*, the order of pictures in the encoded bitstream is: $\overbrace{IPP\cdots PP}^{30}I\cdots$

Since the ME/MC is conducted at both the encoder and decoder, and motion vectors need no transmission, the matching block size in ME/MC is adjustable in the proposed scheme. For a trade-off between computational complexity and motion

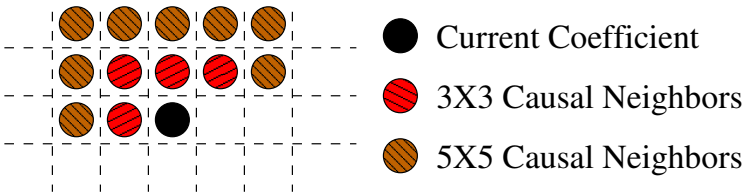


Figure 5.5: Quantized causal neighborhood of current coefficient.

vector accuracy, we chose block size of 4×4 for subbands at each level. The search range at Level n is $[-4^{4-n}, +4^{4-n}]$. The motion vectors are calculated to full-pel accuracy. To reduce the design complexity of the coder, we did not employ MC to the coarsest resolution (highest level) subband. Instead, we simply quantized the subband difference between $\widetilde{LL}_{k,N}$ and $LL_{k+1,N}$.

Fig. 5.5 shows the quantized causal neighbors of the current coefficient. Although both 3×3 and 5×5 neighborhood may be used, we have used a 3×3 neighborhood in our simulation for simplicity. In the design of the optimal normalized quantizer, we choose $n = 65$ as a trade-off between the reconstruction precision and the design complexity. Note that the quantizer table needs to be generated only once and stored as a file. The bin probabilities of each quantizer are also stored as a file. In our simulations, the final bitrate is controlled by changing λ. It has been found that the PSNR values and the bitrates have marginal fluctuation if we keep the same λ for all frames in one GoP.

Fig. 5.6 and Fig. 5.7 demonstrate the efficacy of the proposed ME/MC with LBS technique. It was observed that the proposed technique provides about 2 dB improvement in average over the benchmark scheme without LBS. The incorporation of the LBS technique into backward ME/MC framework boosted the PSNR performance of the reconstructed frames, making the performance comparable to that of the MPEG-2 while supporting multi-level spatial-scalability. Also, it can be seen from the PSNR and the bitrate of the first reconstructed frame that the implemented EQ coder is superior to the MPEG-2 on encoding of the I-frames. It is necessary

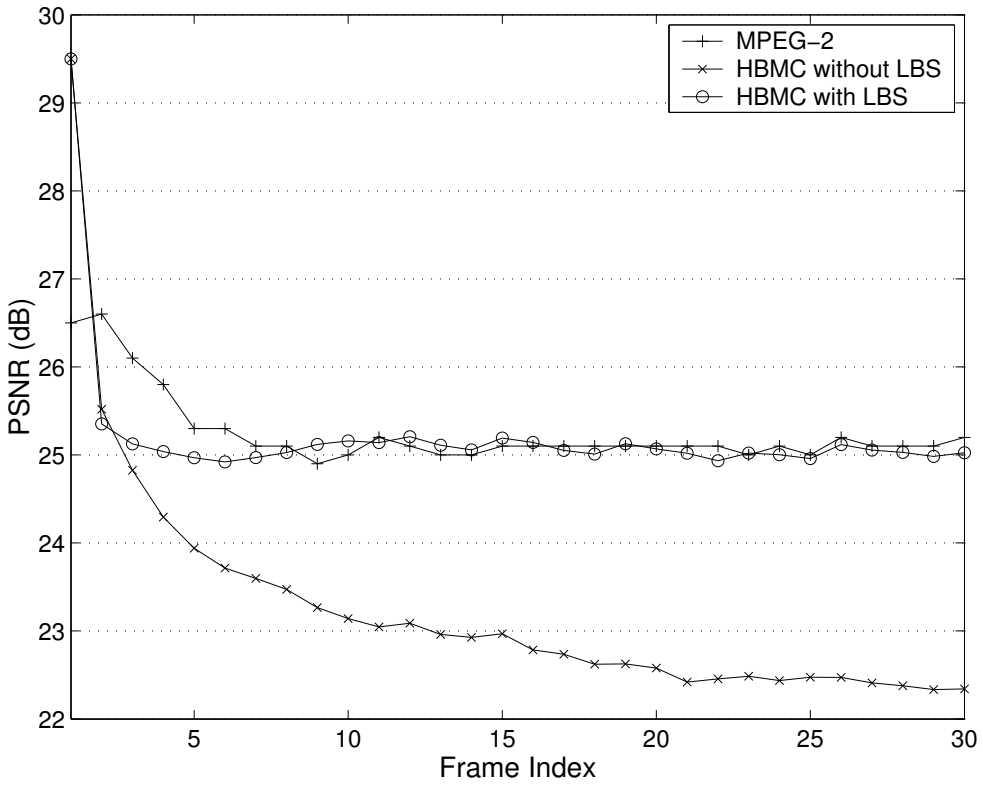


Figure 5.6: Coding results on the *Football* SIF sequence. Comparison of the overall PSNR of the luminance (Y) components at 500 Kbps and 30 fps.

to point out that our coding scheme is still primitive, focusing more on the efficacy test of the proposed technique. There exist many ways, such as forward/backward ME/MC mode switching, overlapped half-pel MC, joint-optimization of ME/MC and residual coding, etc., that when incorporated in the proposed scheme would provide additional improvement.

The original intention of using BMC is that motion information is not explicitly transmitted. Instead, the ME/MC is conducted implicitly in the decoder. Consequently, there is only one type of data, *i.e.*, residual wavelet coefficient, to be transmitted. This can simplify the error-resilience design. Without transmitting motion information, different categories of motion compensation techniques can be tested without considering their impact on bit budget allocation between motion information and residual data. The application of the DWT achieves inherent spatial-

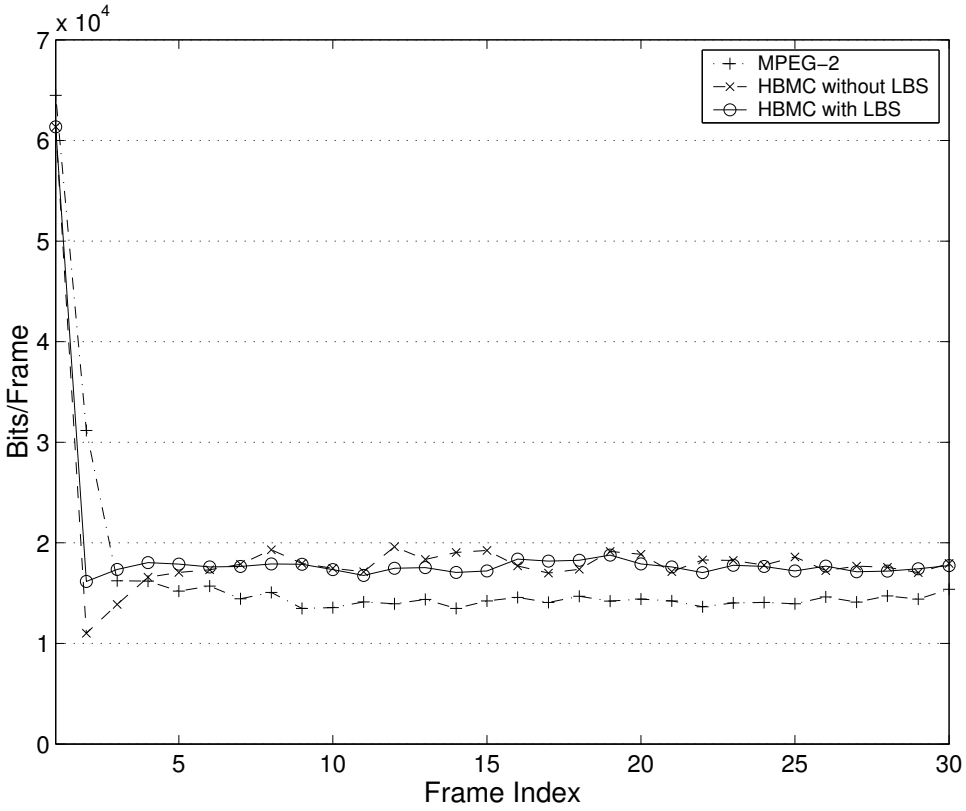


Figure 5.7: Coding results on the *Football* SIF sequence. Comparison of the overall PSNR of the luminance (Y) components at 500 Kbps and 30 fps.

scalability because of its multi-resolution capability. Although these features are highly desirable, it has turned out that the performance of the proposed scheme can only compete with the MPEG-2 after improvements to the basic architecture [141], which is deemed as unacceptable to most researchers. Personally, the author thinks the failure of this architecture is due to the following reasons:

- MC techniques that produce a dense motion field, *e.g.*, pel-recursive and optical flow techniques, failed in the wavelet domain. The author suspects the main reason is that smooth constraint is no longer held in the wavelet domain. This justifies our moving back to a block-matching technique.

- The BMC itself is not efficient at removing the temporal redundancy. Since the prediction of the current frame is based on motion information derived from previous frames, the implication is that the BMC assumes continuous and smooth motion of objects, which does not always hold for natural image sequences.

- The implemented EQ coder [64] performs well on highly-correlated wavelet coefficients, but not on less-correlated residual coefficients.

- The BMC architecture uses a recursive MC and quantization/entropy-coding architecture to satisfy multi-resolution requirement, which is not desirable for robust transmission because of the nested coding dependency among different layers of resolution.

5.2 Wavelet Fine Granular Scalable (WavFiGS) Video Coding

Many wavelet-based compression schemes, such as the EZW [108] and the SPIHT [105], do not work well on motion-compensated residual images. These schemes tend to use

context models such as zerotrees to exploit the statistical dependency across scales within a wavelet pyramid. However, little statistical dependency across the scales of a decomposed residual image is left for exploitation. In addition, the decomposition of a residual image tend to generate many random large-magnitude coefficients in the high-pass subbands, which are difficult for a wavelet image coder to compress.

To handle the inefficiency of wavelet coders on residual images, some variants of wavelet coders have been proposed. For example, ZTE [73] has been proposed to replace the EZW in video coding. The ZTE modifies the EZW to enable explicit quantization and one-pass block coding. The SPIHT has also been modified to facilitate video coding where the SPIHT is employed at the block-level instead of a complete image [60]. The problem with the block-level SPIHT is that rate-allocation among the blocks is difficult since the SPIHT employs implicit quantization and needs an explicit specified-rate for each block bitstream. The common feature of both schemes is that they adopt block-coding strategy to allow flexibility in quantization and content adaptivity. However, both schemes still stick to the zerotree context model, which performance in residual coding is doubtful.

In this section, we proposed a wavelet FGS video coder, which consists of a BL coder and an EL coder. The BL coder is based on the proposed macro-block stack-run (MBSR) coding, which provides similar PSNR performance as that of the H.263 and is superior to the latest proposed hybrid wavelet video coders [84, 142]. The EL coder is based on the modified Golomb RLC to encode the bit-planes of the residual wavelet pyramids. In order to achieve error-resilience, redundancy is taken into consideration in the design of the source coder.

5.2.1 Macro-block Stack-Run Coding

The MBSR is based on the stack-run (SR) coding [123] and designed to be compatible with existing standard coding framework above the MB level. The MBSR only

specifies a method to encode the pels within MBs. It allows any type of ME/MC and rate-control techniques. It depends on other modules to supply residual data and control parameters. In fact, this strategy works so well that we were able to implement the MBSR codec as a plug-in in the TMN H.263 codec [120]. The output bitstream syntax above the MB level is fully compatible with that of the H.263. The only difference is that when the H.263 codec encodes an MB, the MBSR subroutine is called instead.

Context modeling is precluded from the proposed MBSR. In fact, this is exactly why we chose the SR coding as the base for the encoder. Most wavelet image coding schemes take advantage of certain types of context modeling, *e.g.*, the zerotree (parent-children) model and the parent/neighbors model. Although context modeling has been proven to be efficient in the compression of still images, which are normally modeled as slow-varying 2-D statistical fields, its efficacy is doubtful on the motion-compensated residual images, in which the assumption of slow-varying fields no longer holds true. In addition, context modeling often requires usage of special data structures such as trees and lists, which demand extra management of dynamic memory resource and are also inefficient in data accessing.

The MBSR employs block-based coding and explicit quantization to achieve compatibility with standard coders. The block-based coding is desirable because it is good for error-resilience and content adaptivity. In embedded wavelet image coding schemes, coefficients are quantized implicitly using successive implicit quantization. However, the MBSR quantizes the MB coefficients using an explicit step-size so that each MB can be quantized in a content-dependent manner. Note that with explicit quantization, the feature of precise rate-control is not available because only one of the two parameters, *i.e.*, quantizer step-size and output bitrate, can be specified. Here we argue that the precise rate-control is not as important and desirable in video coding because the rate allocation procedure is more complicated with the

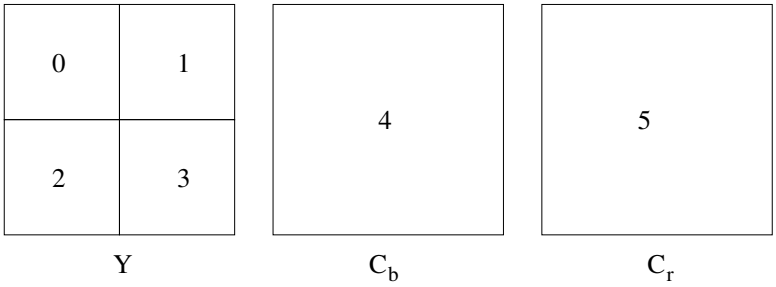


Figure 5.8: The arrangement and indexing of blocks within one MB.

precise rate-control. There do exist many recursive algorithms on optimization of rate-allocation, but application of such algorithms in general applications is infeasible. Furthermore, the rate-allocation among MBs of a frame incurs more difficulty. Another advantage of explicit quantization is that the encoding is completed in one pass instead of multiple passes. Multiple-pass coding does generate an embedded bit-stream, but one embedded bit-stream per MB [60] may not be useful. An embedded bitstream is desirable in scenarios where the bitstream is not expected to be decoded completely. However, since MBs need to be decoded completely in reconstructing an error-free frame, there is no need to produce embedded bitstreams at the MB level. In addition, one-pass encoding is faster than multiple-pass coding.

5.2.1.1 Algorithmic Details of The MBSR

The MBSR is designed to encode MBs of 4:2:0 format raw video sequences. Each MB consists of four luminance (Y) blocks and two chrominance (Cb and Cr) blocks. The size of each block is 8×8. The arrangement and indexing of these blocks within an MB are illustrated in Fig. 5.8.

Before applying the DWT to the MB data, we need to choose among various wavelets. The 9/7-tap bi-orthogonal wavelet [2] is well-known as being optimal for still image coding. However, we have found through experiments that the 9/7-tap wavelet is inferior to the simple Haar wavelet in coding of motion-compensated

residual frames in terms of PSNR quality. Another disadvantage of using wavelets with longer taps is that the "ringing" artifact tends to be more obvious at low bitrates. Hence, the 9/7-tap wavelet was chosen to process intra-coded MBs while the Haar wavelet was applied to inter-coded MBs. When it comes to choosing the level of decomposition, the constraint imposed by the size of a block is obvious. Therefore, a level of 3 was used for all blocks in an inter-coded MB. However, it is well-known that a higher decomposition-level generally brings better quality at the same bitrate. So a level of 4 was used for the 16-by-16 Y component of an intra-coded MB.

Fundamentally, the SR coder is a variant of the one-dimensional (1-D) run-level coders. A scan-order that maps the pair of coordinates of a wavelet coefficient to the index of a linear array is needed just like the zig-zag scan-order used to encode the DCT coefficients. In the proposed MBSR algorithm, the coefficients within a wavelet pyramid are scanned from coarse to fine scales. For the LL and HH subbands, coefficients within each subband are raster-scanned row-by-row. For the HL and LH subbands, coefficients are raster-scanned along the direction of the low-pass filtering.

The aforementioned scan-order only considers the case of an intra-pyramid scanning. However, an inter-pyramid scan-order needs to be specified since the three color components within one MB are transformed separately. A simple solution is to scan and encode the Y, the Cb, and the Cr pyramids separately. However, since the size of the pyramids is small, the overhead of encoding an "end" symbol using AC for each of the six blocks is high. Fig. 5.9 illustrates the inter-pyramid scan-order adopted in our experiments. Different scan orders are used for intra-coded and inter-coded MBs. For the six inter-coded blocks, since coefficients with smaller indices are generally more likely to have a larger magnitude, the scanning for coefficients is fully interleaved. For the three intra-coded blocks, the scanning is done for each block separately because the decomposition levels for the blocks are different and

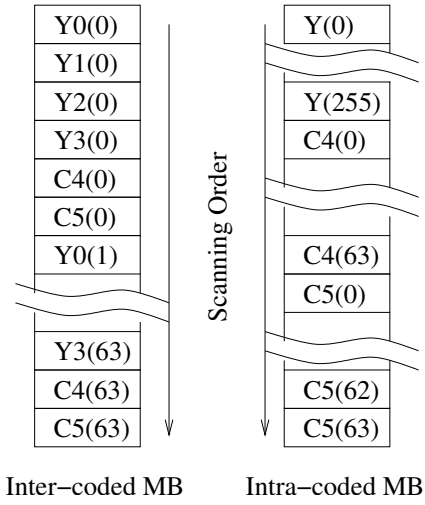


Figure 5.9: The inter-block scan-orders for the intra-coded MB and the inter-coded MB.

our experiments showed that interleaving the scanning process is not helpful.

The coefficients within one MB are scanned and quantized to obtain magnitude (stack) and run-length (run) values. The stack and the run values are then converted to a sequence of symbols, which are chosen from a 4-symbol alphabet. The sequence is finally encoded using a 4-ary AC [123]. The quantizer adopted is a dead-zone uniform quantizer shown in Fig. 5.10, where T is the threshold of the dead-zone and q is the quantization interval of the uniform quantizer. The actual value of q for a specific MB is determined by a rate-control module. The stack value of a coefficient can be calculated as follows using integer arithmetic

$$s(|c|) = \frac{|c|}{2q_s}, \quad q_s \in \{1, 2, \ldots, 31\} \tag{5.10}$$

where c is a wavelet coefficient and q_s is a quantizer scale. It is clear that $q = 2q_s$ and s is always rounded to the smaller integer.

Now the only missing part is the encoding of the mean (or DC) values. In the MBSR, the mean values of the Y, Cb, and Cr blocks are subtracted before DWT is applied to an intra-coded MB. The three mean values are stored in raw 8-bit char

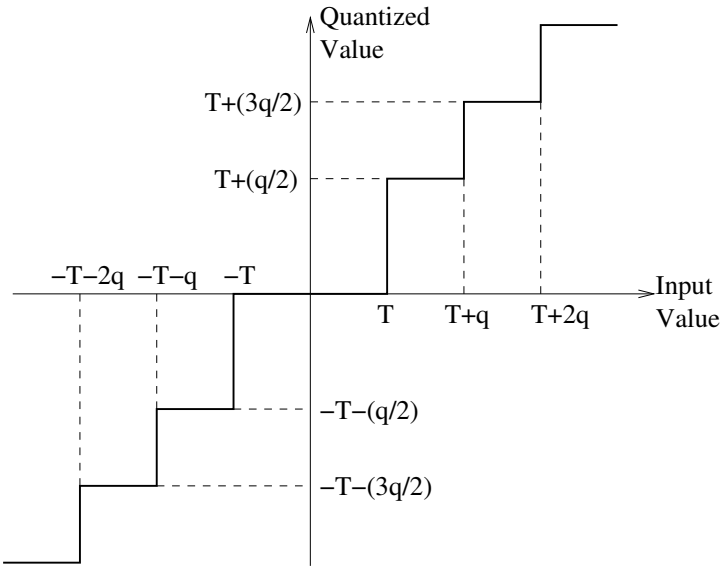


Figure 5.10: The dead-zone uniform quantizer used, where $T = q$.

type. Obviously, this is not optimal since we can encode them with a DPCM coder just like in H.263 [120]. For an inter-coded MB, the pels within the blocks are motion compensated residuals, whose mean values are certain to be around 0. Based on this observation, the mean values of an inter-coded MB are not encoded. In a typical video sequence, since intra-coded MBs normally constitute about 5% of all MBs in an inter-coded frame, encoding the mean values of the intra-coded MBs as raw values is still tolerable, albeit inefficient, in terms of rate budget.

5.2.1.2 Experimental Results

The proposed MBSR coder has been implemented both as a plug-in for the TMN H.263 codec [120] and as the BL coder of the WavFiGS, which employs the PESD in Chapter 4 to encode I-frames and the MBSR to encode the P- and B-frames. The performance of the MBSR in the framework of H.263 is first examined here. The Haar wavelet was used in the decomposition of motion-compensated MBs. The 9/7-tap wavelet was used in the decomposition of intra-coded MBs. In all experiments, only I- and P-frames were used. Since we have not designed a DPCM coder to encode

Table 5.1: Results for encoding 96 frames of the *Carphone* QCIF sequence at 30 fps using different wavelets.

Bit-rate (Kbps)	Wavelet	Average PSNR (dB)		
		Y	U	V
200	Haar	33.79	37.90	38.72
	9/7	33.14	37.55	38.43
500	Haar	38.74	41.31	42.26
	9/7	38.27	40.94	41.85
1000	Haar	43.23	44.73	45.49
	9/7	42.69	44.36	45.06

the mean values of an intra-coded MB, encoding an I-frame using the MBSR will be less efficient. Therefore, we employed the H.263 codec to encode the I-frame (the first frame) when we tested the MBSR. Except for the first frame, the remaining frames in a sequence were encoded as P-frames. The TM-5 off-line rate-control technique was used to set the quantizer scale for each MB [44].

In the experiments that we have conducted, the advanced prediction mode and the unrestricted motion vector mode are both enabled. In Table 5.1, we list the average PSNR results for a typical sequence encoded at different bitrates. We have found that the Haar wavelet consistently achieved better results than did the 9/7-tap wavelet in encoding P-frames.

Fig. 5.11 plots PSNR for Y components using 300 frames of the *Mobile & Calendar* sequence, which contains high activity motion activities.

Table 5.2 compares the MBSR encoding of the sequences (an I-frame followed by all P-frames) to the H.263 encoding. Results for the *Mother & Daughter*, *Container*, *News*, and *Carphone* sequences at bitrates varying from 24 Kbps to 500 Kbps are shown. All the four sequences are in QCIF format (176×144). The first three sequences consist of 300 frames while the *Carphone* sequence consists of 96 frames. These were chosen as representative sequences ranging from slow motion to fast motion. It is shown in Table 5.2 that as the bitrate gets higher, the performance of

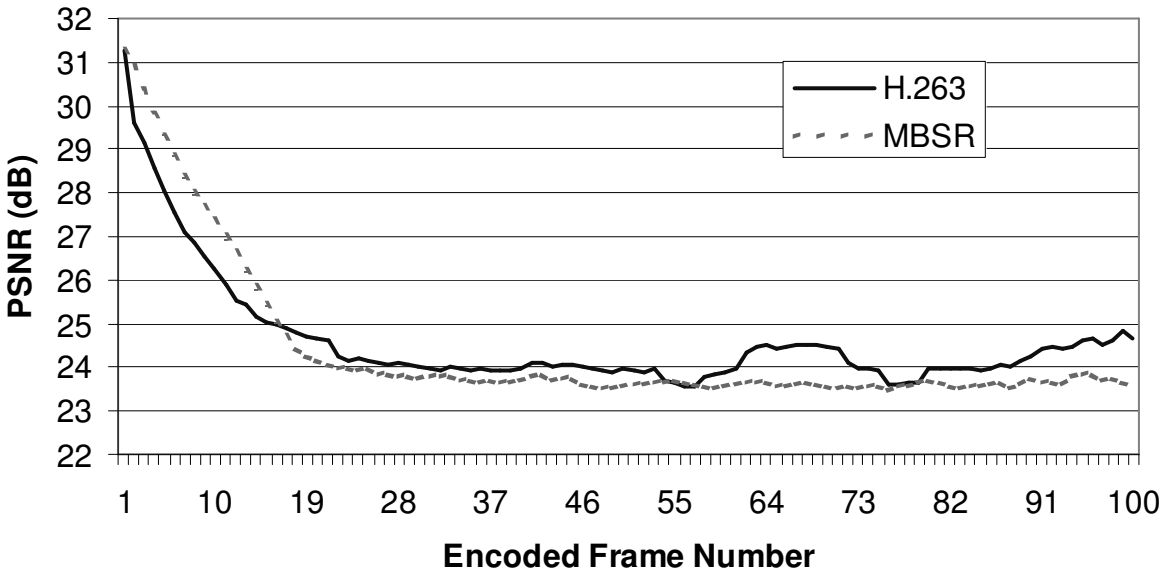


Figure 5.11: Comparison between the H.263 and the MBSR on encoding 300 frames of the *Mobile & Calendar* CIF sequence at 200 Kbps and 10 fps. The PSNR values of the Y component are plotted.

the MBSR exceeds that of the H.263 in terms of PSNR. At very low bitrates, the MBSR is slightly inferior to the H.263 in PSNR while the subjective quality is quite similar as recorded in the reconstructed video. Although the average PSNR of the MBSR does not exceed that of the H.263 under all testing conditions, it does perform much better than the zerotree-based hybrid video codecs. It has been reported that an optimized EZW-based technique can achieve a gain of 0.7-0.8 dB over MPEG-2 in bitrates ranging from 500 Kbps to 1.5 Mbps [142]. However, the MBSR achieves a gain of about 2.7 dB over the MPEG-2 under the same testing conditions. This result clearly favors the MBSR over the optimized EZW-based technique.

It is worth noting that the TM-5 rate-control technique, under these test conditions, was able to meet most target bitrates with an average error of 0.5% for both the MBSR and the H.263. However, for extreme low and high bitrates, the average error reached up to 5% for both schemes. The actual bitrate was always higher (lower) than the extreme low (high) target bitrate for both schemes.

Note that the results discussed above were obtained in the H.263 framework,

Table 5.2: The average PSNR results over entire sequences. Bitrate is in Kbps.

Clip	Bit-rate (Kbps) & fps	Codec	Average PSNR (dB)		
			Y	U	V
Mother & Daughter	24@7.5	H.263	35.78	40.63	41.38
		MBSR	34.71	39.59	40.69
	250@30	H.263	41.10	45.48	46.04
		MBSR	41.39	44.63	45.34
	500@30	H.263	44.58	46.53	47.07
		MBSR	45.22	47.27	47.88
Container	48@7.5	H.263	36.26	41.05	40.65
		MBSR	35.53	39.87	39.68
	250@30	H.263	40.51	44.67	44.52
		MBSR	40.20	44.02	43.87
	500@30	H.263	43.79	47.44	47.50
		MBSR	44.09	47.69	47.77
News	48@7.5	H.263	34.69	38.46	39.05
		MBSR	33.63	37.09	38.10
	250@30	H.263	41.11	44.12	44.46
		MBSR	40.30	43.03	43.36
	500@30	H.263	44.83	46.77	47.27
		MBSR	45.89	47.78	48.11
Carphone	48@7.5	H.263	32.90	37.84	38.50
		MBSR	32.09	37.09	37.93
	250@30	H.263	35.79	39.73	40.44
		MBSR	34.82	38.57	39.46
	500@30	H.263	39.14	42.03	42.79
		MBSR	38.74	41.31	42.26

i.e., B-frames were disabled and there was only one encoded I-frame. In order to design a wavelet FGS BL coder that has similar GoP structures as the MPEG series standards, the PESD in Chapter 4 and the MBSR were combined. The WavFiGS BL coder employs the PESD to encode I-frames and the MBSR to encode P- and B-frames. The bitstream syntaxes and ME/MC of the MPEG-2 standard specified in [41] were employed. Fig. 5.12 illustrates the PSNR performance of the the two MBSR variants compared with the other two benchmark coders, namely, the MSSG MPEG-2 codec [80] and the BBC's latest Dirac wavelet codec [84]. The Dirac codec is also based on hybrid-coding architecture, incorporating techniques such as variable-size block matching ME/MC, OBMC, and AC of wavelet coefficients. The 300 frames of the *Foreman* sequence were encoded at 350 Kbps and 30 fps. There are several patterns noticeable in Fig. 5.12. The first pattern is that there are spikes in the PSNR results when a wavelet still-image coding scheme is used in I-frame coding. The spikes show up periodically in the results of the WavFiGS and the Dirac, which can be attributed to that the DWT is superior to the DCT in still-image coding. The second pattern is that the H.263 with the MBSR plug-in performs better than the WavFiGS. Since the two schemes employ the same MB coding method, this difference in PSNR performance shows the improvements gained from employing better compression of motion/header information and better prediction techniques, such as OBMC and variable-size ME/MC , by the H.263 over the MPEG-2. It is also observed in Fig. 5.12 that the improvement of the MBSR as the H.263 plug-in over MPEG-2 is fairly big. Note that the last two huge values of the Dirac were caused by bugs in its implementation. The last two frames were simply left uncoded by the Dirac.

Fig. 5.13 compares the visual quality of a reconstructed frame. The frame is the 100th frame extracted from the reconstructed sequences. It is easily perceived that Fig. 5.13 (b) provides similar quality as of Fig. 5.13 (a) except around the mouth

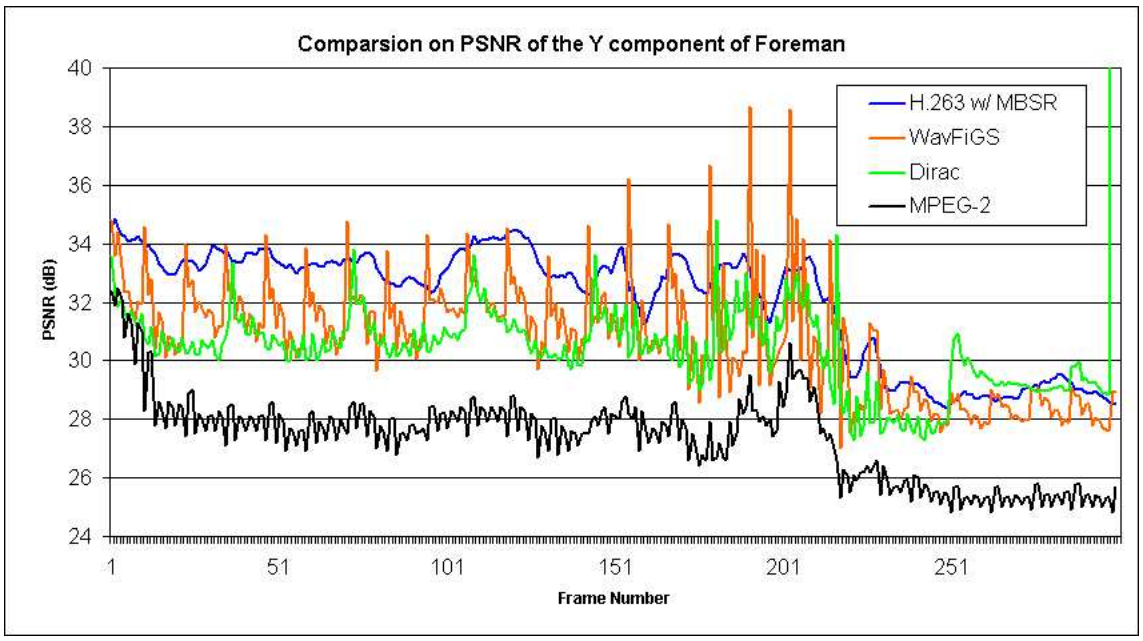


Figure 5.12: The comparison on the PSNR quality of the four schemes. The 300 frames of the *Foreman* sequence are encoded at 350 Kbps and 30 fps.

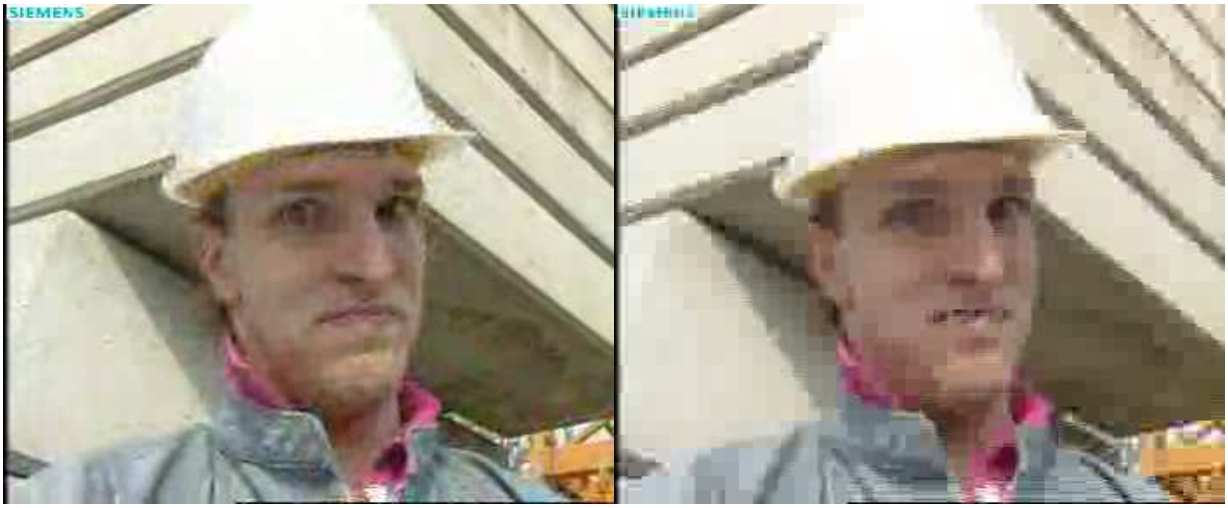
of the person, where blocky artifacts are still perceptible. It is reasonable to believe that the better ME/MC techniques in the H.263 handle ME/MC better as seen in Fig. 5.13 (a). Overall, Fig. 5.13 (c) looks fuzzier than Fig. 5.13 (a) and Fig. 5.13 (b), and the color of Fig. 5.13 (c) is inferior to the other three sub-figures, which implies an inferior coding performance of the Dirac on the chrominance components. The worst sub-figure is definitely Fig. 5.13 (d) where block artifacts show up globally, clearly exhibiting that the bit budget assigned to the frame is not enough to achieve a good reconstructed frame.

5.2.2 Error Resilient Enhancement Layer Coding

As had been discussed in §2.3, there are several categories of techniques that can be employed to provide error-resilience to a compressed image and video. Roughly speaking, the techniques can be classified into two categories from another perspective, those explicitly insert redundancy and those implicitly provide redundancy. The

(a) H.263 with MBSR plug-in

(b) WavFiGS



(c) Dirac

(d) MPEG-2

Figure 5.13: The reconstructed 100th frame of the *Foreman* sequence in display order, which is encoded as a P-frame.

resynchronization marking technique belongs to the former category since markers are explicitly inserted into a bitstream. The MDC technique belongs to the latter category because no redundant information is explicitly inserted when each single description is encoded. However, the overall coding performance is decreased by encoding multiple complementary descriptions. The performance loss can also be interpreted as caused by the implicit redundancy insertion.

In our design, we are more interested in techniques that provide redundancy implicitly. This type of error-resilience is typically called *compartments strategy*, because one broken compartment cannot sink a vessel. Similarly, the overall quality of an image or video will be fine if only a small part of it is erroneous. In fact, this strategy has been proposed in robust wavelet image coding. Creusere proposed a robust EZW coder [21] in which wavelet coefficients representing the same spatial area are clustered and encoded independently. In other words, a zerotree is encoded separately from the coding of other zerotrees. During transmission, the loss of one zerotree will not affect the decoding of any other received zerotree bitstreams. In another robust EZW coder [103] for packet-oriented scenarios, a zerotree is encoded independently and the output of the zerotrees are packed into fixed size datagrams so that the loss of one packet has no effect on the decoding of other received packets.

5.2.2.1 Error Resilience from Localization and Decoupling

In packet-oriented networks, source data are fragmented into datagrams before they are encapsulated by underlying communication protocols. A well-established principle for data fragmentation is *application layer framing* (ALF) [17]. According to the ALF principle, packets must be constructed consistent with the structure of a source coding algorithm. In error-resilient coding for packet networks, research has focused on how to alleviate the effects of packet loss. In the BL coding, the effects of packet loss are more serious because of the temporal dependency involved in inter-frame

coding, *e.g.*, the loss of one packet for an I-frame tends to influence all the following frames in the same GoP. The good news in the FGS EL coding is that the residual data in the EL are intra-encoded without inter-frame coding, which means that errors in an EL frame will not propagate to other frames. Further on, an encoded EL frame may span over a couple of packets, and an important goal of coding is to prohibit the effects of a packet loss from propagating to other received packets. This can be called the *decoupling strategy*. The slice structure used in the MPEG series standards is another good example of error resilient techniques, in which a slice only represents a small part of the frame. This method can be called the *localization strategy*. Both strategies are employed in the proposed EL coder.

As far as it is understood by the author, the idea of FGS originated from progressive still image coding, specifically, the embedded wavelet image coding schemes that involve the concept of bit-plane coding, which was first proposed by Shwartz and Baker [112]. The concept of progressive bit-plane coding is general enough that DCT coefficients can also be encoded with it. In fact, the progressive bit-plane coding is the basic idea underlying the MPEG-4 FGS profile [42]. In the MPEG-4 FGS EL coder, the 64 DCT coefficients of an 8×8 residual block are zig-zag scanned into an array and organized into multiple binary sequences with each binary sequence representing one bit-plane. The statistical properties of different binary sequences are then analyzed to design a universal variable-length code-set to encode the so-called (RUN, EOP) pairs, where RUN represents distance between two consecutive significant bits, "1", and EOP indicates whether it is the last significant bit in the bit-plane.

To encode a residual wavelet pyramid after the MBSR coding of P- and B-frames and the PESD coding of I-frames, elementary Golomb (EG) codes [29, 85] have been adopted due to their flexibility when the input source statistics are entirely unknown *a priori* and are subject to frequent changes. A basic assumption made

here is that the probability p of the run symbol satisfies $p > 0.5$, which is almost always true. If it is false, stop symbol and run symbol can be swapped and the analysis is still valid. For a positive integer parameter m, which satisfies $p^m = 0.5$, let EG_m denote a variable-to-variable length code defined over the extended alphabet $\{1, 01, 001, \ldots, 0^{m-1}, 0^m\}$, where 0^ldenotes a sequence of l zeros. Using the EG_m, the symbol 0^m is encoded with a 0, whilst 0^l1, $0 \leqslant l < m$, is encoded with a 1 followed by the binary representation of l (using $\lfloor \log_2 m \rfloor$) bits if $l < 2^{\lceil \log_2 m \rceil} - m$ and $\lceil \log_2 m \rceil$ bits otherwise [86]. The EG_m is equivalent to a Golomb code [33] of order m when applied to a concatenation of extended input symbols. However, the difference does exist in that a Golomb code is defined for infinite alphabet while an EG_m is defined over a finite alphabet for better adaptation.

In the design of the EL coder, localization strategy is adopted to divide a frame into slices. The size of the Y (luminance) component of a slice is:

$$w_{sl} = w_{fl}, \quad h_{sl} = 16 \tag{5.11}$$

where w_{sl} denotes the width, h_{sl} denotes the height, and w_{fl} denotes the frame width of the luminance component. Similarly, the size of the chrominance components of a slice is:

$$w_{sc} = w_{fc}, \quad h_{sc} = 8 \tag{5.12}$$

The total number of slices, N_s, can be expressed as:

$$N_s = \frac{h_{fl}}{h_{sl}} \tag{5.13}$$

where h_{fl} is the height of the frame.

Each bit-plane of a slice is encoded into a bitstream. Ideally, the size of a compressed bit-plane of a slice should be no bigger than that of the path MTU. In our

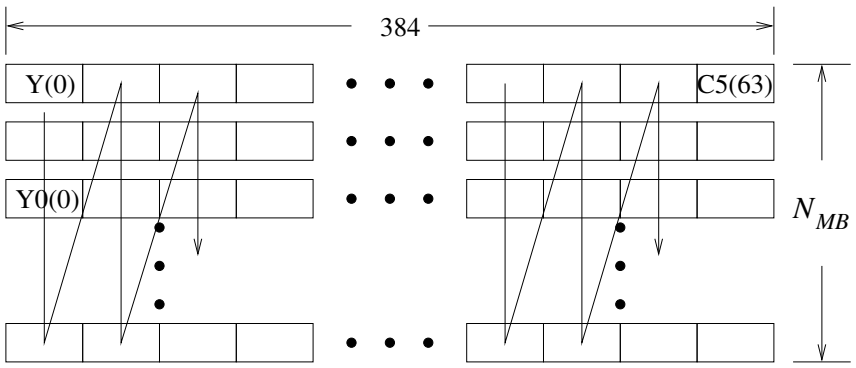


Figure 5.14: The scan order of the coefficients of macro-blocks within one slice.

experiments with CIF sequences, the size of the compressed finest bit-planes does not exceed 800 bytes. However, it is expected that such size of compressed bit-planes will exceed size of the path MTU when high-definition sequences are encoded. As far as we are concerned, we assume that every compressed bit-plane can be transmitted in one packet without further fragmentation.

For I-frames, the wavelet coefficients representing a slice are scanned in the same scan-order as in Chapter 3. For P- and B-frames, each MB within a slice is decomposed separately. In order to apply the scan-oder depicted in Fig. 5.9 on a slice, the first coefficients in the MBs are scanned first, followed by the second coefficients in the MBs, and so on. The basic idea is to scan the low-pass coefficients before the high-pass coefficients as illustrated in Fig. 5.14, where the number of MBs within one slice, N_{MB}, can be expressed as:

$$N_{MB} = \frac{w_{sl}}{16} \tag{5.14}$$

The total number of coefficients in one slice is $384 \cdot N_{MB}$, which means that in each bit-plane we need to encode that many bits. Algorithm 5 contains the pseudo-codes of the RLC algorithm we used to encode the binary bit-planes, which is a modified version of the MELCODE [85]:

The bit-plane encoder traverses through the slice following the scan-order. The

Algorithm 5 Error resilient slice bit-plane encoder using elementary Golomb codes.

```
/*S is a pointer to the input slice*/
/*b is an integer indicating current bit-plane*/
/*b_min is an integer indicating the finest bit-plane*/
bit_plane_encoder(S,b,b_min) {
    num_bits := b − b_min; k :=T[I]; m := 2^k;
    run_length := 0;
    for i := 1 to length(S) do {
        if(S_b[i] == 0) {
            run_length++;
            if(run_length == m) {
                output_bit(1);
                I ← min{I_max, I + 1};
                k := T[I]; m := 2^k; run_length := 0;
            }
        }
        else {
            output_bit(0);
            output_bits(run_length, k); /*k bits for run_length*/
            output_bit(sign(S[i]));
            value := shift_right(S[i], b_min); /*shift value right by b_min bits*/
            output_bits(value, num_bits);
            S[i] := S[i] - shift_left(value, b_min);
            I ← max{0, I − 1};
            k := T[I]; m := 2^k; run_length := 0;
        }
    }
}
```

bit of each coefficients at bit-plane b, $S_b[i]$, is checked to determine whether the coding should be in a *run* mode or a *stop* mode. If the bit has a value of 0, the coding is in the run mode and the run-length counter is increase by one. If the maximum run-length m is reached in the run mode, the encoder assumes that current estimate of p is not accurate and m is adjusted according to the function $T[I]$, which is defined as:

$$T[I] = \begin{cases} \lfloor I/4 \rfloor & \text{if } 0 \leqslant I < 16 \\ \lfloor I/2 \rfloor - 4 & \text{if } 16 \leqslant I < 24 \\ I - 16 & \text{if } 24 \leqslant < 32 \end{cases} \tag{5.15}$$

Note that if $T[I] = I$, the bit-plane encoder will function as the adaptive run-length coder proposed by Langdon [29]. In our experiments, equation (5.15) made the adaptation process more robust and less susceptible to spurious estimates than directly setting $k = I$.

When the encoder is in the stop mode, the least significant k bits of the run-length counter are outputted followed by the sign bit. The purpose of this step is to encode the position of the significant coefficient and its sign. The following step provides error-resilience. Suppose we want to encode a value of 30, which is 11110_2 in binary format. If we know that the finest bit-plane encoded is bit-plane 1, we output "111". If the the finest bit-plane encoded is bit-plane 0, we output "1110". The most significant bit is already encoded when the run-length is encoded. With this method, the final magnitude of a coefficient is determined once it is significant and the decoupling among packets is realized.

If we do not want to incorporate error-resilience in the bit-plane encoder, the magnitude of a coefficient is progressively refined when more bit-planes are available. The problem with such a bit-plane encoder is that if any bit-plane is erroneous, all finer bit-planes will be rendered useless, which certainly does not conform to

the decoupling strategy. Pseudo-codes of such a bit-plane encoder without error-resilience is listed in Algorithm 6.

Algorithm 6 Slice bit-plane encoder without error resilience using elementary Golomb codes.

```
/*S is a pointer to the input slice*/
/*b is an integer indicating current bit-plane*/
/*b_min is an integer indicating the finest bit-plane*/
bit_plane_encoder(S,b,b_min) {
    num_bits := b − b_min; k :=T[I]; m := 2^k;
    run_length := 0;
    for i := 1 to length(S) do {
        if(S_b[i] == 0) {
            run_length++;
            if(run_length == m) {
                output_bit(1);
                I ← min{I_max, I + 1};
                k := T[I]; m := 2^k; run_length := 0;
            }
        }
        else {
            output_bit(0);
            output_bits(run_length, k); /*k bits for run_length*/
            if(S[i] encoded first time) output_bit(sign(S[i]));
            |S[i]| := |S[i]| − 2^b;
            I ← max{0, I − 1};
            k := T[I]; m := 2^k; run_length := 0;
        }
    }
}
```

The complete architecture of the WavFiGS encoder is shown in Fig. 5.15.

5.2.2.2 Experimental Results

It should be noted here that no specific rate-control algorithm has been implemented in the WavFiGS EL coder. The bitrate and the PSNR results were sampled when bit-planes of the residual wavelet pyramids were encoded completely. In the benchmark MPEG-4 FGS coder [77], the rate-control for the EL is not implemented, either. The sample values of the MPEG-4 FGS coder were taken when bit-planes of the residual

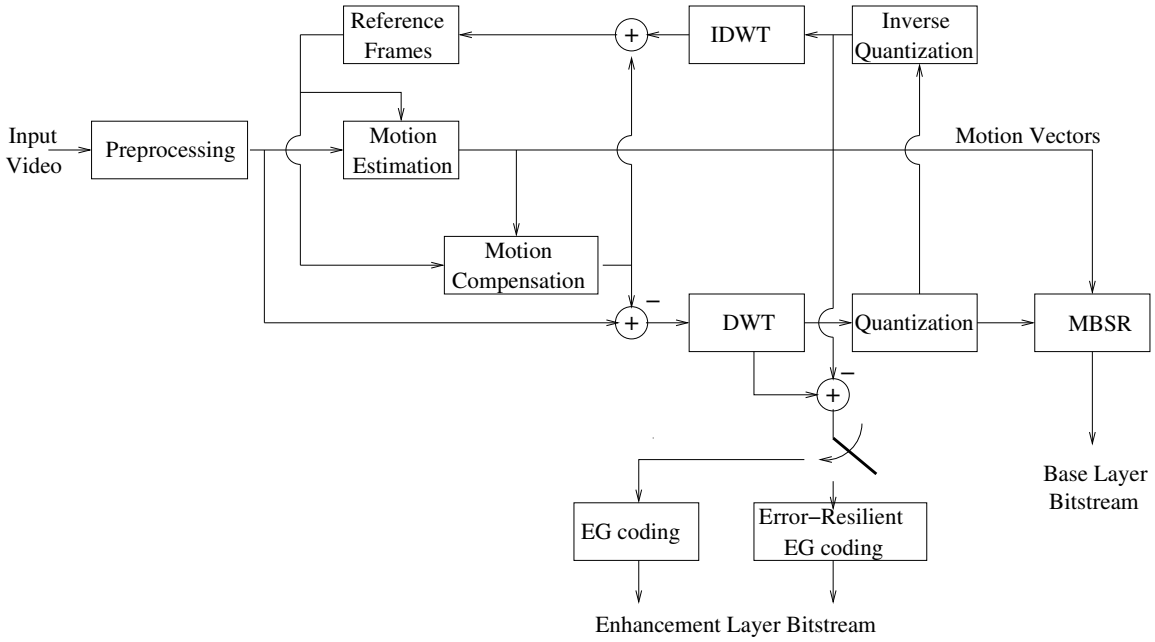
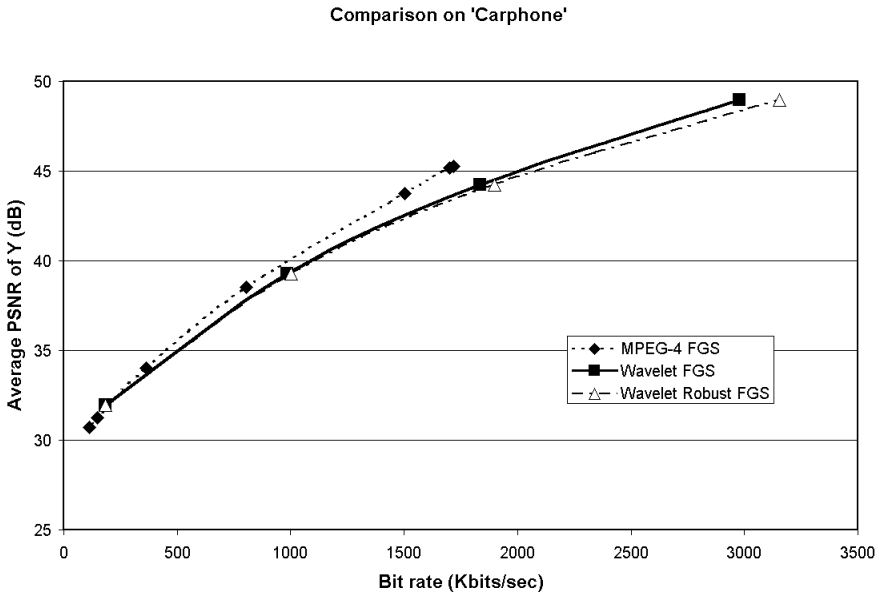


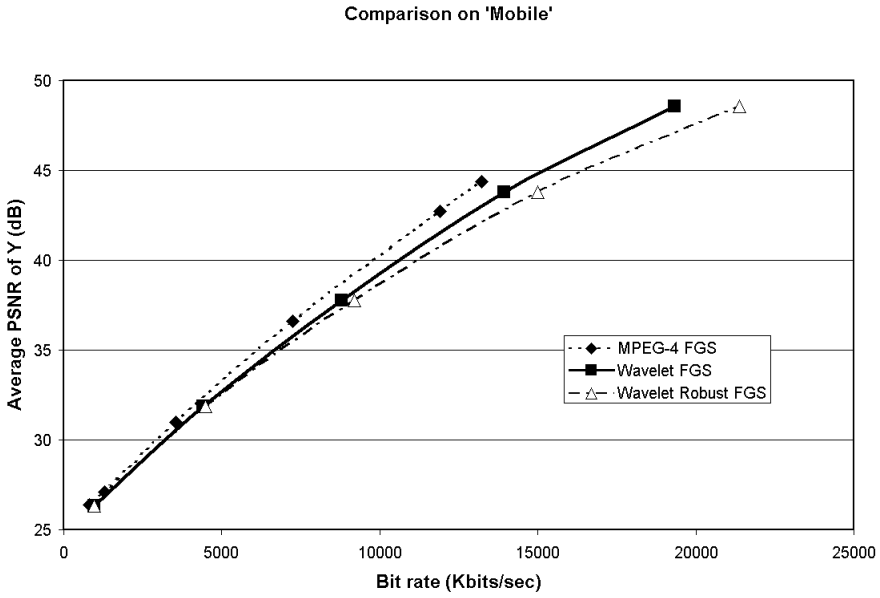
Figure 5.15: The complete architecture of the WavFiGS encoder.

DCT coefficients were encoded completely. It is shown that the performance gap is fairly small below the 40 dB threshold. Since a PSNR value of 40 dB corresponds to a mean square error of about 6.5, the fairly bigger performance gap above 40 dB does not mean much difference in the reconstructed pel values. In fact, 1 dB PSNR gap above 40 dB threshold only indicates that the difference in MSE is about 1 or less. This said, the curves in Fig. 5.16 should be interpreted as meaning that the three schemes provided similar coding performance. Also, Fig. 5.16 illustrates that the error-resilient WavFiGS EL coder does not cost significantly more bit-budget. The comparison on the performance of video streaming by employing these two FGS EL coders will be discussed in Chapter 6.

The WavFiGS encoder generates two meta-data files after encoding a sequence. The format of the meta-data file for the BL is the same as shown in Table 6.3. The format of the meta-data file for the EL is shown in Table 5.3, where "f0" denotes FGS layer for Frame 0. The size of each bit-plane of a slice is recorded and will be used in streaming the EL.



(a) Carphone



(b) Mobile

Figure 5.16: The average PSNR of Y component results obtained from encoding two sequences by the WavFiGS EL coder.

Table 5.3: Format of the WavFiGS EL source trace file.

Frame Number	Frame Type	Bit Plane	Slice Index	Length (bytes)
f0	I	4	0	107
f0	I	4	1	70
f0	I	4	2	47
f0	I	4	3	51
f0	I	4	4	38

5.3 Summary

We focused on the application of wavelets in scalable video coding in this chapter. We first proposed a wavelet-based spatial scalable video coder that can inherently achieve multi-level spatial scalability, *i.e.*, a new level of finer resolution is automatically achieved when it is synthesized by the next lower level resolution image (an low-pass subband) and the corresponding enhancements (the three high-pass subbands). This spatial scalable video coder generates motion information through backward motion compensation, which means that no transmission of explicit motion information is needed. In the second section, we discussed the design of a wavelet-based robust FGS video coder for usage in packet oriented environment. We adopted the decoupling strategy and the localization strategy to achieve robustness of the generated bitstream. The output of the enhancement layer encoder is formed into small source packets that can be flexibly organized into network transport packets. Each source packet is to be indenpendently decoded to atomically increase the reconstructed video quality.

Chapter 6

Forward Error Correction in Packetized Video Streaming

Due to the lossy characteristic of today's Internet, we need certain mechanisms to handle the transmission errors (in the form of packet loss) in packetized video streaming. In Chapter 5, we have discussed the design of an error-resilient FGS coder. Note that the BL of a compressed FGS video is encoded to achieve the highest possible compression ratio. The task of providing protection for a BL bitstream is not shouldered by the source coder. In this chapter, we investigate the employment of FEC techniques for protection of compressed video data. An analytical model of FEC for video streaming is established in §6.1. Comparison with another analytical FEC model is presented in §6.2. Streaming of an error-resilient FGS video with FEC is discussed in §6.3.

6.1 FEC Performance in Video Streaming

FEC techniques are the preferred error-control schemes for multicast or interactive streaming applications. Traditionally, the FEC techniques are used in a generic way that does not consider the inherent characteristics of the transmitted data [102, 104].

With the advent of scalable video streaming, the concept of layered FEC has been proposed to facilitate different levels of protection for the layers of a video [57, 116]. Later, researchers have been investigating the inherent temporal-dependency within the compressed video data, such as the prediction dependency existing among the three types of encoded frames, to maximize the performance of the FEC [74, 138]. Based on the analytical models, the three types of the encoded frame, *i.e.*, the I-, P-, and B-frames, are allocated different FEC coding rates based on their influence on the transmission-error propagation. However, the FEC can be applied to video data in more than one way. Before a thorough study on different FEC application strategies is completed, it cannot be claimed that the strategies proposed in [74, 138] are necessarily the best strategies. The analysis in this section builds upon the work on an analytical model for the MPEG video proposed in [138]. From now on, we will called the model established in [138] as *frame-based* FEC technique since different types of frames receive different levels of FEC protection. We will show by our analysis and simulation results that applying FEC techniques at the group of picture (GoP) level, dubbed *GoP-based* FEC, achieves better protection than frame-based FEC under typical packet loss conditions.

The class of linear erasure codes discussed in [100] is modeled to provide FEC for video source datagrams in our simulation. The linear erasure codes are so called because they can be analyzed using the properties of linear algebra. An (n, k) linear erasure code encodes k source data items into n $(n > k)$ transport data items. If at least k out of n transport data items are successfully delivered, the original k source data items will be decoded. Otherwise, none of the lost data items is recovered. Let $\mathbf{x} = \{x_0, x_1, \ldots, x_{k-1}\}$ denote the source data and $\mathbf{G}$ denote an $n \times k$ generator matrix, an (n, k) linear erasure code can be expressed as

$$\mathbf{y} = \mathbf{G}\mathbf{x} \tag{6.1}$$

where any sub-matrix $\mathbf{G}'$ made of k rows from $\mathbf{G}$ is invertible. If the identity matrix $\mathbf{I_k}$ is one of such sub-matrices, the transport data items include the source data items and the code is called a *systematic code*. The advantage of the systematic code is that no channel-decoding process is virtually needed when the probability of erasure is sufficiently small.

Let $\mathbf{y}'$ denote a sub-vector that consists of k elements from $\mathbf{y}$ and $\mathbf{y}*$ the remaining $(n-k)$ elements of $\mathbf{y}$. Assume $\mathbf{y}'$ is delivered to the receiver, we have the following relationship

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}' \\ \mathbf{y}* \end{bmatrix} = \begin{bmatrix} \mathbf{G}' \\ \mathbf{G}* \end{bmatrix} \mathbf{x} \tag{6.2}$$

where $\mathbf{G}'$ consists of k rows from $\mathbf{G}$ that corresponds to $\mathbf{y}'$. Since we know that $\mathbf{G}'$ is invertible by the definition of a linear erasure code, we can recover the source data items by using

$$\mathbf{x} = \mathbf{G'}^{-1}\mathbf{y}' \tag{6.3}$$

6.1.1 Parameters and Variables

The following parameters are treated as fixed attributes of a streaming video flow for the duration of the streaming session:

- t_{RTT}: the round-trip time
- t_{RTO}: the TCP retransmission time-out interval
- s_{pkt}: the packet size in bytes
- s_I: the average size of I-frames (in packets)
- s_P: the average size of P-frames (in packets)

- s_B: the average size of B-frames (in packets)
- N_P: the number of P-frames within a GoP
- N_{BP}: the number of B-frames between immediately successive P-frames
- K: the average total number of source datagrams in a GoP.
- N: the average total number of source and redundant datagrams in a GoP.

Only the packet loss probability p will be treated as a variable. It is clear from the definition above that

$$K = s_I + s_P \times N_P + s_B \times (N_p + 1) \times N_{BP} \tag{6.4}$$

and

$$N - K \geq 0 \tag{6.5}$$

The effect of erasure codes on the transmission of video packets is modeled as a series of Bernoulli trials. The probability $q(N, K, p)$ that K source datagrams are successfully transmitted with $N-K$ redundant datagrams over a lossy network with packet loss probability p is

$$q(N, K, p) = \sum_{i=K}^{N} \left(\begin{array}{c} N \\ i \end{array} \right) (1-p)^i p^{N-i} \tag{6.6}$$

The analytically derived TCP-friendly sending rate is given in [88] as

$$T = \frac{s_{pkt}}{t_{RTT}\sqrt{\frac{2p}{3}} + t_{RTO}\left(3\sqrt{\frac{3p}{8}}\right) p(1+32p^2)} \tag{6.7}$$

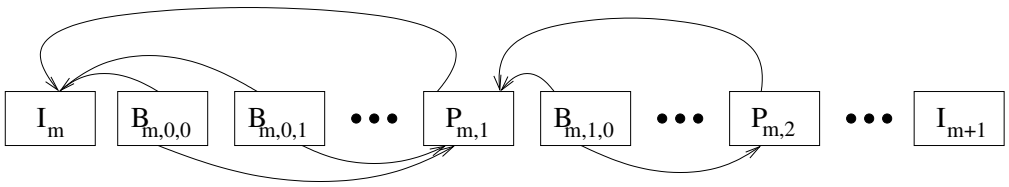


Figure 6.1: The structure of the m-th GoP and the inter-frame dependency relationship within it.

6.1.2 Playable Frame Rate

The term *playable frame rate* (PFR) is coined in [25] to evaluate the video streaming performance. It is defined as the expected number of frames received error-free in one second. If a GoP structure is fixed, the rate of GoP per second can be analytically expressed since the size of a GoP in packets is known. After the GoP rate is determined, the frame rate of I-, P-, and B-frames can be determined through the frame dependency relationship. The GoP structure and the inter-frame dependency are illustrated in Fig. 6.1. Note that the first N_{BP} B-frames within the m-th GoP are dependent on the last P-frame within the $(m-1)$-th GoP. This implies that as an exception, we have only $(1+N_P(N_{BP}+1))$ frames in the first GoP of a sequence.

The GoP rate is computed as

$$G = \frac{T}{N \cdot s_{pkt}} \tag{6.8}$$

It is worth noting here that G denotes the *playable* GoP rate instead of source GoP rate. In fact, source GoP rate is fixed once the source frame rate and the GoP structure are known. The source GoP rate, G_s, can be expressed as

$$G_s = \frac{f}{(1+N_P)(1+N_{BP})} \tag{6.9}$$

where f is the source frame rate, which usually takes a value of either 25 or 30. Whenever $G \geqslant G_s$ is satisfied, it can be said that the network condition is good enough

to support error-free streaming. Otherwise, we may experience quality degradation during the streaming process.

Since there is only one I-frame within a GoP, the playable rate of I-frames is

$$R_I = G \cdot q_I \tag{6.10}$$

where q_I is the probability of successfully delivering all packets belonging to an I-frame.

Before we try to compute q_I, we classify the delivery of N packets into three situations using the number of lost packets L:

1. The I-frame is playable when $L \leq N - K$.
2. The I-frame is probably playable when $N - K < L \leq N - s_I$.
3. The I-frame is not playable when $L > N - s_I$.

Therefore, q_I is computed as

$$q_I = \sum_{i=0}^{N-K} \binom{N}{i} p^i (1-p)^{N-i} + \sum_{i=N-K+1}^{N-s_I} \binom{N-s_I}{i} p^i (1-p)^{N-i} \tag{6.11}$$

The first P-frame, P_1, can only be displayed when its preceding I-frame and itself are successfully transmitted. Similarly, we can have following situations for P_1:

1. P_1 is playable when $L \leq N - K$.
2. P_1 is probably playable when $N - K < L \leq N - s_I - s_P$.
3. P_1 is not playable when $L > N - s_I - s_P$.

Therefore, q_{P_1} is computed as

$$q_{P_1} = \sum_{i=N-K+1}^{N-s_I-s_P} \binom{N-s_I-s_P}{i} p^i(1-p)^{N-i} + \sum_{i=0}^{N-K} \binom{N}{i} p^i(1-p)^{N-i} \quad (6.12)$$

The playable rate of P-frames is

$$R_P = G \cdot q_P \quad (6.13)$$

where $q_P = \sum_{i=1}^{N_P} q_{P_i}$. Now, we can calculate q_P as

$$\begin{aligned} q_P &= \sum_{m=1}^{N_P} \sum_{i=N-K+1}^{N-s_I-ms_P} \binom{N-s_I-ms_P}{i} p^i(1-p)^{N-i} \\ &\quad + N_P \cdot \sum_{i=0}^{N-K} \binom{N}{i} p^i(1-p)^{N-i} \end{aligned} \quad (6.14)$$

If we label the I-frame in a GoP as I_0, the P-frames as P_i, where $1 \leq i \leq N_P$, and B-frames as $B_{i,j}$, where $0 \leq i \leq N_P$ and $0 \leq j < N_{BP}$. All B-frames in the same interval between an I- and a P-frame (or two P-frames) have the same playable frame rate, which can be expressed as

$$q_{B_{i,0}} = q_{B_{i,1}} = \cdots = q_{B_{i,N_{BP}-1}} \quad (6.15)$$

For all B-frames except those immediately preceding the I-frame of the next GoP,

the probability of successful delivery for $B_{i,j}$ is

$$q_{B_{i,j}} = \sum_{m=N-K+1}^{N-s_I-(i+1)s_P-s_B} \left(\begin{array}{c} N-s_I-(i+1)s_P-s_B \\ m \end{array} \right) p^m(1-p)^{N-m} + \sum_{m=0}^{N-K} \left(\begin{array}{c} N \\ m \end{array} \right) p^m(1-p)^{N-m} \quad (6.16)$$

For the B-frames immediately preceding the I-frame of the next GoP, we have the probability for $B_{N_P,j}$ as

$$q_{B_{N_P,j}} = q_I \cdot \left(\sum_{m=N-K+1}^{N-s_I-N_Ps_P-s_B} \left(\begin{array}{c} N-s_I-N_Ps_P-s_B \\ m \end{array} \right) p^m(1-p)^{N-m} + \sum_{m=0}^{N-K} \left(\begin{array}{c} N \\ m \end{array} \right) p^m(1-p)^{N-m} \right) \quad (6.17)$$

The playable rate of B-frames is

$$R_B = G \cdot q_B \quad (6.18)$$

where q_B can be expressed as

$$q_B = N_{BP} \cdot \sum_{i=0}^{N_P} q_{B_{i,0}} \quad (6.19)$$

The total PFR is

$$R = R_I + R_P + R_B \quad (6.20)$$

6.2 Non-Scalable Video Streaming Using FEC

In this section, the closed-form formula derived in §6.1 is compared with that of the frame-based FEC model in [138]. The PFR computed using the two models will be compared in §6.2.1. In §6.2.2, we will use the non-scalable MPEG-4 trace [26] and the NS-2 network simulator [95] to develop modules and conduct simulations on FEC in video streaming.

6.2.1 Model-Based Analysis

We first describe the methodology used. The network and video stream settings used in the formulae are then discussed. The section is completed by an analysis of the results obtained from applying the analytical models.

To study the effects of different FEC strategies on streaming performance for a given packet loss probability, p, we execute the following steps to compare the resultant PFR, R.

1. Given a specified p, compute the TCP-friendly sending rate T using equation (6.7).

2. Compute R using equation (6.20) and the equations given in [138].

3. Treat FEC settings, t_{RTT}, and s_{pkt} as tunable parameters. For each and every combination of these three parameters, compute and plot the PFR using the equations derived in §6.1 and in [138].

The network settings, such as packet size, RTT, and loss characteristics, are taken from typical network connections [90]. As in [27], t_{RTO} is set as $t_{RTO} = 4t_{RTT}$. Note that the bitrate of streamed videos is highly diversified, it could range from 64 Kbps to 10 Mbps. In our model-based analysis, the bitrate is set at 1.15 Mbps (MPEG-1 VCD quality). As observed in [55], each type of the I-, P-, and B-frames uses about

Table 6.1: Network settings.

t_{RTT} (msec)	25, 50
t_{RTO} (msec)	100, 200
s_{pkt} (bytes)	500, 1000
p (%)	0.5, 1, 2, ..., 10

1/3 of the total bit budget when a typical GoP structure ($N_P = 3$, $N_{BP} = 2$) is employed. Although this estimate cannot reveal the statistical nature of the size of the compressed frames, it does simplify the analysis and more emphasis could be placed on the FEC and the comparison of the two models. The parameter values used in the models are depicted in Table 6.1.

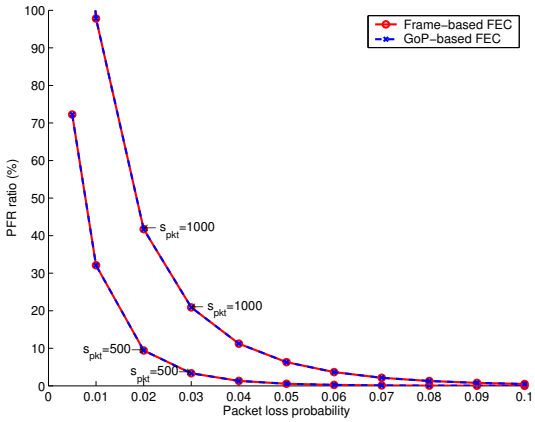
The results generated by the close-form formulae are plotted in Fig. 6.2. In each sub-figure, the two models as well as the effect of packet size on the performance are compared. First let us compare the effect of packet size on the streaming performance. We only compare two packet sizes, $s_{pkt} = 500$ and s_{pkt} =1000. Note that to stream the the same video clip, the total number of source datagrams is roughly doubled in the former situation. To keep a constant channel-coding rate, we also double the number of redundant datagrams when $s_{pkt} = 500$. Although the channel-coding rate is identical in both cases, the streaming performance with $s_{pkt} = 1000$ is significantly better. This result can be explained by equation (6.7). If we drop the numerator, s_{pkt}, from the equation, we can see that the equation actually tells us how many packets could be sent in a TCP-friendly way in one second. In other words, equation (6.7) only specifies how fast we can send packets in a TCP-friendly way. When we employ smaller packet size, the packet rate has to be increased to match the bitrate of the streamed video, in other words, more packets need to be sent in a unit time and the limitation set by equation (6.7) is met earlier. This explains why the performance of using smaller packet size is almost always inferior in all the sub-figures of Fig. 6.2 and Fig. 6.3. The difference between these two figures is that

the t_{RTT} of Fig. 6.2 doubles that of Fig. 6.3, which implies smaller bandwidth.

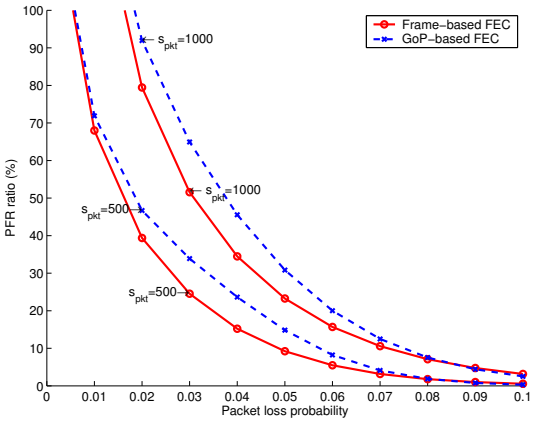
In Fig. 6.2 (a), the setting for the frame-based FEC is (0, 0, 0), which means there is no FEC available. Since there is no FEC in the streaming, the performances of the frame-based FEC and the GoP-based FEC are identical. The PFR deteriorates very quickly when the stream is not FEC-protected. We can see that only around 40% of the frames can be delivered error-free at $p = 0.02$, which is fairly small. After we provide a light-weight FEC, (1, 1, 0), the PFR reaches near 100% at the same value of p. Here we use (s_{IF}, s_{PF}, s_{BF}) to denote the allocation of redundant datagrams among different frame types, *i.e.*, s_{IF} packet(s) for I-frames, s_{PF} packet(s) for P-frames, and s_{BF} packet(s) for B-frames. The corresponding number of packets, R, for the GoP-based FEC under the same channel-coding rate is

$$R = s_{IF} + N_P \cdot s_{PF} + (N_p + 1) \cdot N_{BP} \cdot s_{BF} \tag{6.21}$$

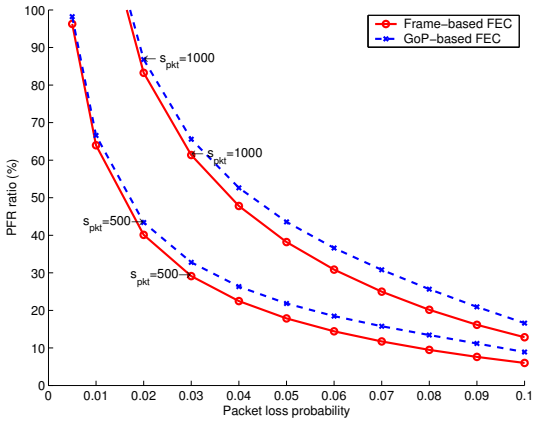
An interesting phenomenon is that at the same probability, $p = 0.02$, providing heavy-weight FEC, *e.g.*, (8, 4, 1), results in worse PFR performance. The reason is that the FEC incurs bandwidth overhead. If the FEC provided exceeds an appropriate level, it occupies unnecessary extra bandwidth that could have been used to transmit source datagrams. This mismatch between the FEC level and the value of p is illustrated in Fig. 6.4. Note that since the case of $s_{pkt} = 500$ always results in inferior performance, we consider only $s_{pkt} = 1000$ here. In Fig. 6.4 (a), we can see that the total number of source datagrams is near 150 whilst 50 redundant datagrams are needed when the heavy-weight FEC, (8, 4, 1), is employed. At $p = 0.02$, the network only supports the transmission of about 150 packets per second in a TCP-friendly way. When we try to use heavy-weight FEC, about 50 packets end up not being transmitted by the streaming server. This results in an inferior performance as compared to light-weight FEC, (1, 1, 0), where almost all packets can be sent out. In Fig. 6.4 (b), the network supports transmission of about 210 packets a second at



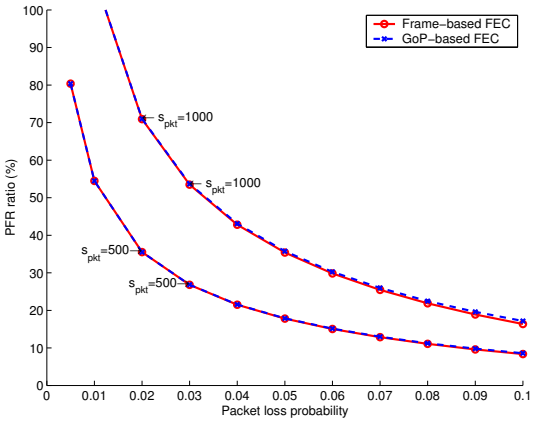
(a) Frame: (0, 0, 0), GoP: 0



(b) Frame: (1, 1, 0), GoP: 4

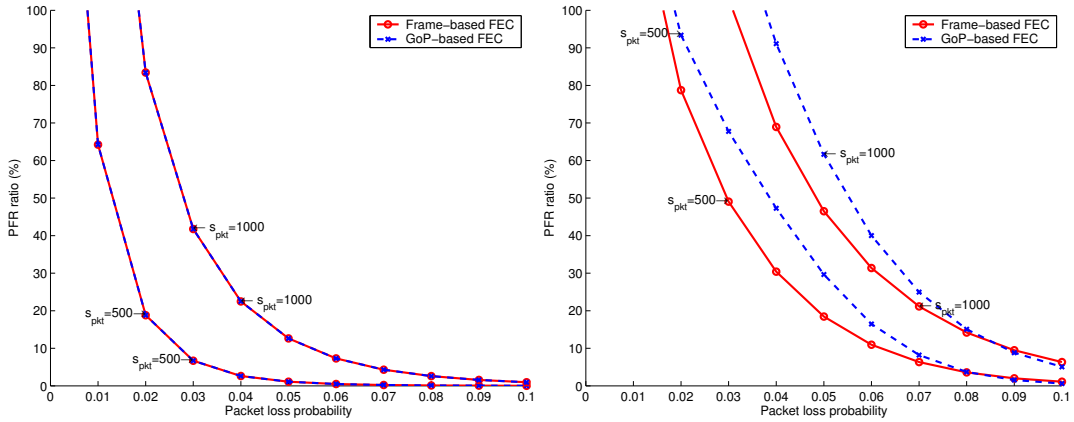


(c) Frame: (4, 2, 0), GoP: 10



(d) Frame: (8, 4, 1), GoP: 28

Figure 6.2: Comparison on ratio of PFR vs. source frame rate when $t_{RTT} = 50$ ms and $t_{RTO} = 200$ ms.



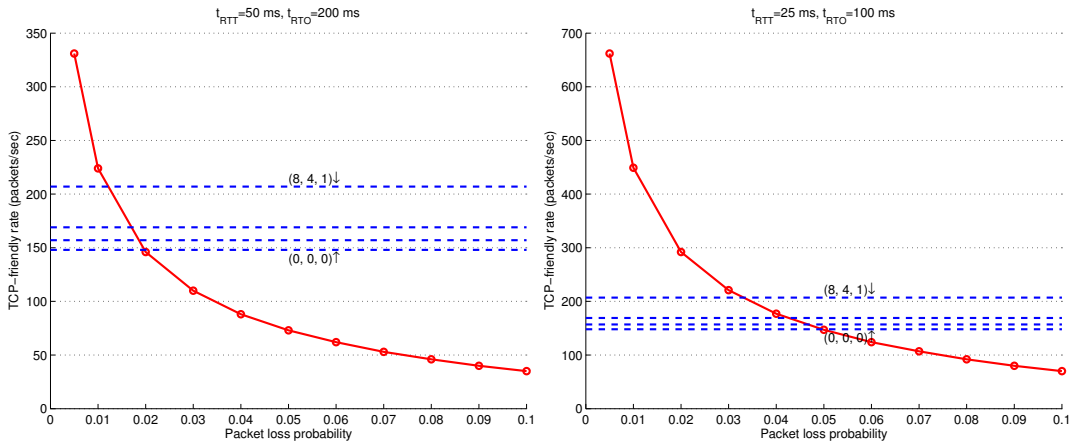
(a) Frame: (0, 0, 0), GoP: 0

(b) Frame: (1, 1, 0), GoP: 4

(c) Frame: (4, 2, 0), GoP: 10

(d) Frame: (8, 4, 1), GoP: 28

Figure 6.3: Comparison on the ratio of PFR vs. source frame rate when $t_{RTT} = 25$ ms and $t_{RTO} = 100$ ms.



(a) (b)

Figure 6.4: The TCP-friendly transmission packet rate vs. the packet rate demanded by FEC-protected streaming applications.

$p = 0.03$. In this case, all packets will be sent out and the resulted PFR is well above the source frame rate at $p = 0.03$, as shown in Fig. 6.3 (d).

From the analysis above, we know that heavier FEC protection does not necessarily lead to better streaming performance since the overall bandwidth is limited. To demonstrate that the FEC settings should be adjustable to reach an optimal point for different network loss probabilities, Fig. 6.5 is plotted. Fig. 6.5 (a) depicts thirteen combinations of FEC settings meant to be used under various network loss probabilities. In the other three sub-figures, these thirteen combinations are tested for three sets of loss probability. It is shown in these sub-figures that the two models achieve their respective highest PFRs at similar FEC settings. In fact, the GoP-based FEC schemes does provide better highest PFR performance under all situations.

Although the GoP-based FEC almost always performs better than the frame-based FEC in a practical setting, we do observe that the GoP-based FEC is computationally more demanding than the frame-based FEC. In the frame-based FEC, three types of (n, k) settings, namely, $(s_I + s_{IF}, s_I)$, $(s_P + s_{PF}, s_P)$, and $(s_B + s_{BF}, s_B)$

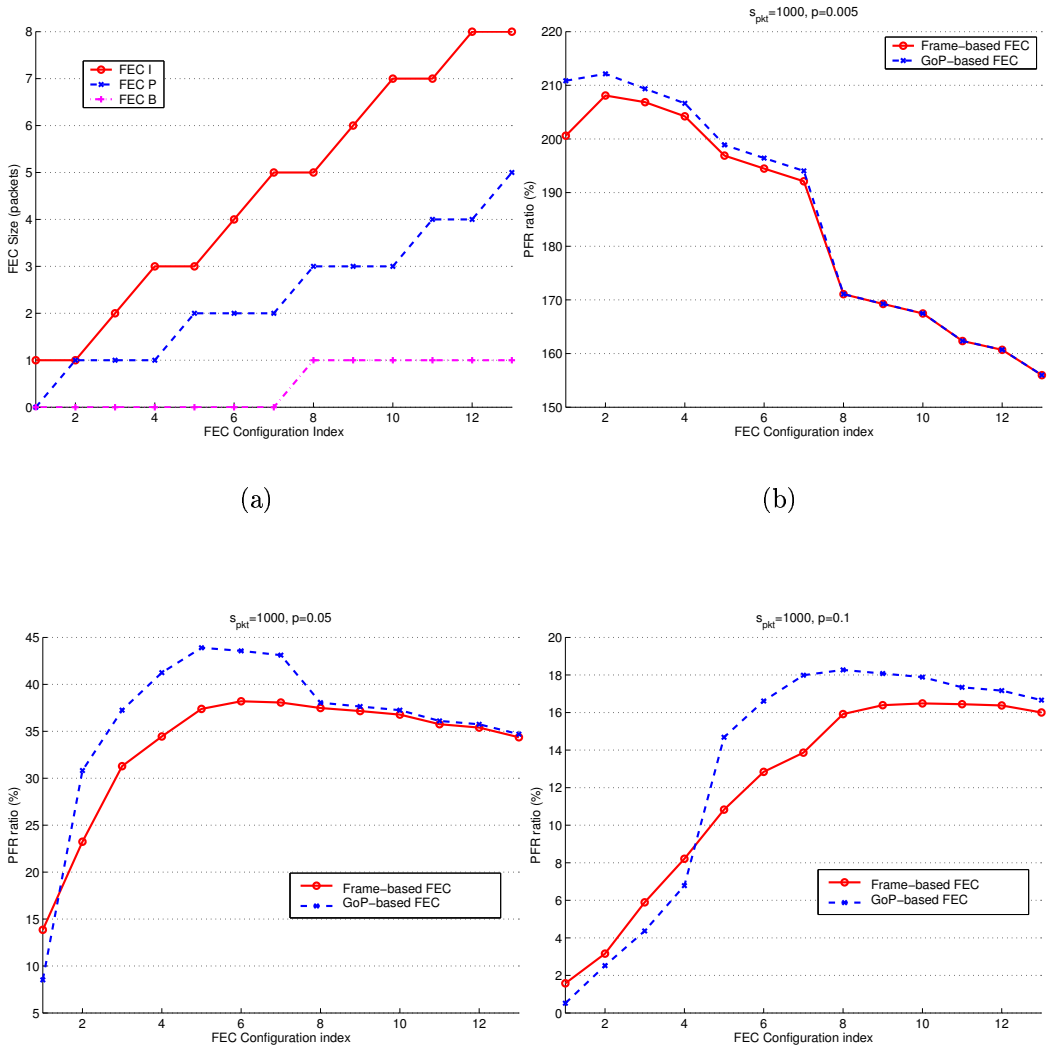


Figure 6.5: Adjusting FEC configuration to search for the best one under specific network loss probability.

are used. In the GoP-based FEC, only one (n, k) setting is used. Since the erasure codes we model here are systematic codes, only the redundant packets need to be generated. By employing a generator matrix $\mathbf{G}$ defined in equation (6.1), we need s_I multiplications and one summation to generate one byte within a redundant datagram for I-frames in the frame-based FEC. Since we have s_{IF} redundant datagrams for I-frames, totally we need $s_{pkt} \cdot s_{IF} \cdot s_I$ multiplications and $s_{pkt} \cdot s_{IF}$ summations for I-frames. Since s_{pkt} is only a proportional factor and multiplication is generally more time-consuming than summation, we drop s_{pkt} and summation in our further analysis.

In the frame-based FEC, the number of multiplications we need for generating redundant datagrams is

$$M_f = s_{IF} \cdot s_I + N_P \cdot s_{PF} \cdot s_P + (N_P + 1) \cdot N_{BP} \cdot s_{BF} \cdot s_B \tag{6.22}$$

while the number in a GoP-based FEC is

$$M_g = (s_{IF} + N_P \cdot s_{PF} + (1 + N_P) \cdot N_{BP} \cdot s_{BF}) \cdot (s_I + N_P \cdot s_P + (1 + N_P) \cdot N_{BP} \cdot s_B) \tag{6.23}$$

The relative complexity can be shown by the complexity ratio, c_r

$$c_r = \frac{M_g}{M_f} \tag{6.24}$$

Table 6.2 depicts the relative complexity of GoP-based FEC compared to frame-based FEC. Since the values of frame size are average values, the c_r values we get are also estimated values. The change of c_r under different situations is fairly small. Since the generation of redundant datagrams makes up only a small part in the computation of streaming applications, we do not expect a significant impact on overall computational complexity by replacing a frame-based FEC technique with a

Table 6.2: The relative complexity when generating redundant packets with $s_I = 24$, $s_P = 8$, and $s_B = 3$.

s_{IF}	s_{PF}	s_{BF}	c_r
1	1	0	6
4	2	0	5
8	4	1	6.46

GoP-based FEC technique.

6.2.2 Simulation-Based Analysis

We have studied the performance of the two models using closed-form formulae in §6.2.1. However, it is obvious that parameters used in the analysis are fixed values while real network conditions and compressed frame sizes are statistical at various temporal scales. Ideally, an analysis on streaming performance should be conducted on traces generated by streaming sessions in real networks. But design and implementation of a real-time streaming system should be a team effort and we are more interested in certain aspects of the video streaming application. To obtain a reasonable abstraction, we used the NS-2 [95] as our simulation platform and developed add-on modules for our own study.

Instead of using a fixed mean compressed frame size to compute the PFR, we used the real trace files [26] of non-scalable videos generated by an MPEG-4 encoder, which are downloadable from `http://www-tkn.ee.tu-berlin.de/research/trace/trace.html`. It is worth noting here that there exists an MPEG-4 traffic generator for the NS-2 that is downloadable from `http://www.sce.carleton.ca/~amatrawy/mpeg4`. However, we anticipate that a real trace file is better for simulation provided it is available. Table 6.3 shows the first five lines of a typical trace file. Note that the frames are numbered in the encoding order while their number in the raw sequence can be determined from the column of the display time.

Table 6.3: Format of the MPEG-4 source trace file.

Frame Number	Frame Type	Display Time (ms)	Frame Size (bytes)
1	I	0	1117
2	P	120	780
3	B	40	475
4	B	80	578
5	P	240	368

To study the effects of FEC techniques on video streaming performance under a controllable and comparable situation, we did not employ the usual multiple-node-with-one-bottleneck-link topology. Instead, we set up only two nodes connected by a duplex link. One node functions as a streaming server while the other one functions as a client. A configurable error model is inserted between the server and the client, so that the packet loss probability, p, is controllable and results from different FEC strategies could be compared. The streaming server reads entries from a trace file, generates source datagrams, and passes the source datagrams to the UDP agent for transmission. Since the trace file represents a variable bitrate compressed video, the client contains a receiver buffer to smooth out the bitstream variation. The decoder then periodically accesses the receiver buffer to retrieve packets for decoding. If all packets for a frame and its reference frame(s) are received, the frame is labeled playable except otherwise. The PFR is determined by the ratio of the playable frames versus all frames sent by the server.

In a typical simulation session, the system set-up is unambiguously specified by an OTcl script used in the NS-2. The following excerpt written in OTcl depicts an example of the network set-up in our simulation:

```
...
#Create two nodes
set n0 [$ns node]
```

```
set n1 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
set em [new ErrorModel]
$em set rate_ 0.05
$em unit pkt

$em drop-target [new Agent/Null]
set rv [new RandomVariable/Uniform]
set rng [new RNG]
$rng seed 0
$rv use-rng $rng
$em ranvar $rv
$ns link-lossmodel $em $n1 $n0

#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set packetSize_ 1000
set udp1 [new Agent/UDP]
$udp1 set packetSize_ 1000
$ns attach-agent $n1 $udp1
$ns connect $udp0 $udp1

set tfile [new VideoTracefile]
$tfile filename Verbose_DieHardIII_10.dat
```

```
$tfile set seed_ 123456

set server [new Application/Traffic/VideoTrace/FEC]
$server attach-agent $udp1
$server attach-tracefile $tfile
$server set frameRate_ 25
$server set enableFEC_ 1
$server set NumGOPs_ 1
$server FEC cfg-file fec.cfg
$server FEC config Frame

set client [new Application/VideoDecoder/FEC]
$client attach-agent $udp0
$client set enableFEC_ 1
$client set NumGOPs_ 1
$client set bufferedTime_ 10
$client set bitRate_ 2Mb
...
```

In order to streaming video between a server and a client, two nodes and a duplex link are constructed. To flexibly adjust the network loss probability, a built-in error model is inserted between the two nodes. This error model, *$em*, is controlled by a random variable, *$rv*, with uniform distribution. Note that other error models can also be used instead. Two UDP agents are initialized to handle the packets and are attached to the two nodes. An MPEG-4 trace file is then opened for access. The starting frame in this MPEG-4 trace file is determined by the member *seed_* of *$tfile*. A streaming server is then instantiated as *$server*, whose parameters are set accordingly. The configuration of the FEC is read from a configuration file, *fec.cfg*.

This configuration file contains the (n, k) pairs for all three frame types. Either the frame-based FEC or the GoP-based FEC can be chosen. The number of packet blocks is specified by *NumGOPs_*, which must keep consistent between the server and the client. The client is instantiated as *$client*, in which the video bitrate and the buffered time must be specified to let us calculate the size of the receiver buffer in unit of packets. Note that this size of receiver buffer is used when no FEC is involved in the streaming. When FEC is employed, the buffer size in the unit of packets can be inferred from the (n, k) pairs.

In our simulation, a 10-minute clip was streamed out of the movie "Die Hard III", which is encoded at medium quality using an MPEG-4 encoder and downloadable at `http://www-tkn.ee.tu-berlin.de/research/trace/pics/FrameTrace/mp4/index3c01.html`. Independent packet loss events during a streaming session were assumed throughout our simulations. Although many would argue that practical packet loss events are bursty rather than independent as found in [90, 62], we propose to interleave packets within GoPs during transmission so that the assumption of independent lost packets is still valid. We will further discuss this part in §A.1. The simulation results with different FEC configurations are illustrated in Fig. 6.6. The settings of the physical link information is shown in the excerpt. For each and every value of the network loss probability, we used ten different seed values for the random number generator to generate different packet loss patterns. Note that in Fig. 6.6, we only plot the mean values of the PFR under different packet loss patterns. For comparison clarity, the same curve that corresponds to streaming without FEC is plotted in every sub-figure.

As clearly shown in Fig. 6.6, the GoP-based FEC scheme always performs better than the frame-based FEC scheme when an optimal FEC configuration is considered. Just as what we have predicted using the derived analytical model, the frame-based scheme performs better than the GoP-based scheme when the packet loss becomes

worse. For example, in Fig.6.6 (a) the frame-based scheme beats the GoP-based scheme when the packet loss probability exceeds 0.12. However, by our analytical model the FEC configuration (1, 1, 0) is only optimal near a packet loss probability of 0.005. When the packet loss probability exceeds 0.12, heavier FEC protection should be employed and the GoP-based scheme performs better again.

In results obtained from simulations with real trace files, two important factors are different from what we have discussed in §6.2.1: (1) the compressed frame size in bytes and packets varies instead of being fixed, and (2) a streaming buffer is involved to smooth out the frame size variation. The first factor has important implication on how FEC is employed. The second factor determines whether there will be losses due to small buffer size. Since we are more interested in the effects of FEC on streaming performance, we simply make the buffer large enough to eliminate any packet loss due to buffer size. In our simulation, we found a 10-second buffering is enough for the clip we chose.

In the analytical models, the mean size of the three frame types are used in calculation. However, this is not the case in simulation involving a real source trace. For example, the maximum frame size in our clip is 8161 bytes and the minimum frame size is 26 bytes. After packetization, the maximum frame size is 9 packets and the minimum frame size is 1 packet if we set the datagram length to be 1000 bytes. Although the mean size of B-frames is 3 packets, the biggest B-frame in our clip needs 6 packets. Hence, if we provide one redundant datagram for each B-frame and each packet belonging to the B-frame is protected, the actual channel-coding rate for the B-frames varies from 6/7 to 1/2. This implies that the bigger the frame is, the less protection it gets. Although this result is far from optimal, keeping a constant channel rate across all individual B-frames are too costly and impractical because the generator matrix $\mathbf{G}$ needs to be recomputed each time the (n, k) pair is changed. Adjusting FEC configuration frequently will cause too much system overhead and is

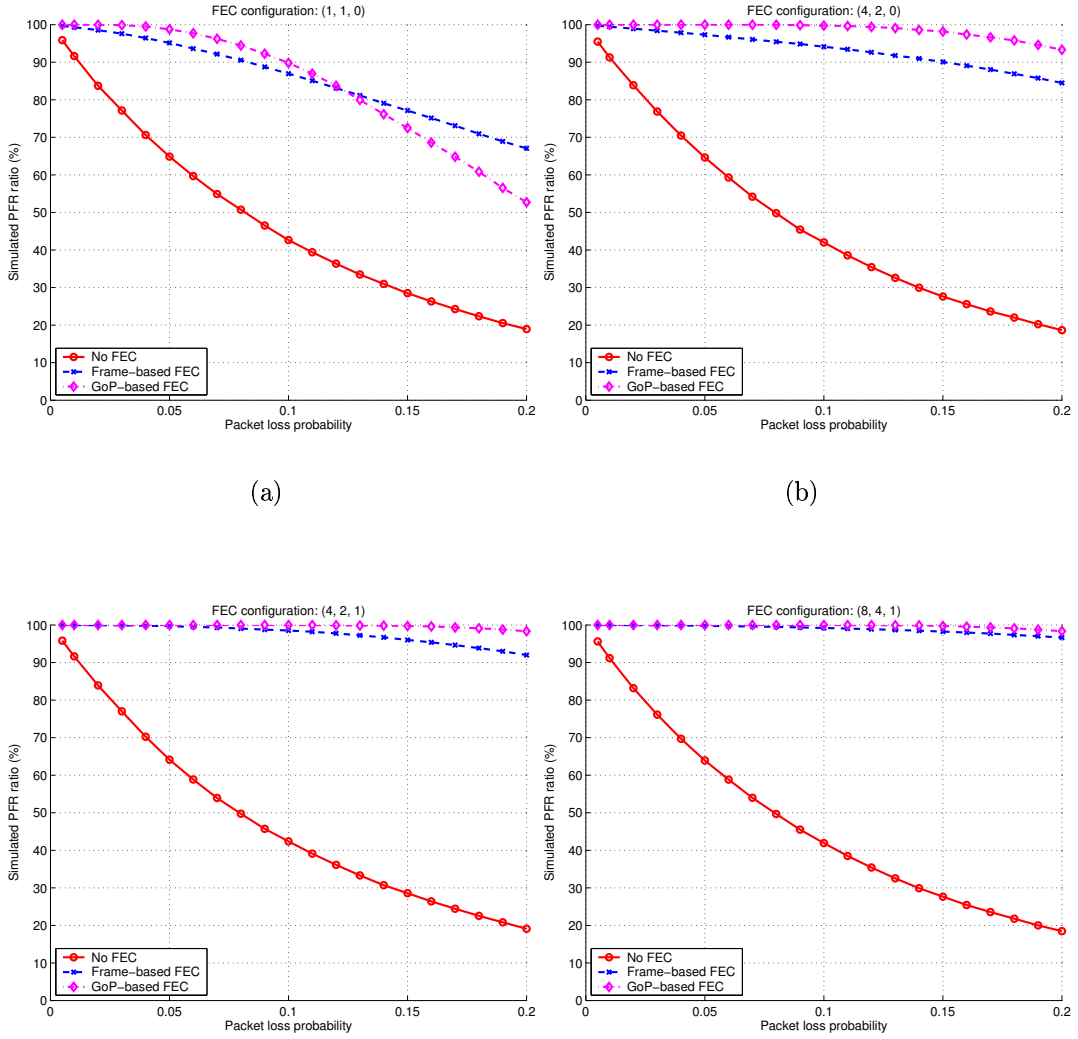


Figure 6.6: The effect of adjusting FEC configuration on the performance of video streaming using an MPEG-4 non-scalable source trace.

not suggested. However, keeping the same FEC configuration for the duration at the temporal scale of hours is obviously not desirable either because the frame size tends to change drastically when there are scene changes in the video. The optimized FEC configuration adjustment during streaming could use some future research work.

6.3 FGS Video Streaming Using FEC

Employing FEC to a compressed video provides certain-level of error protection with the costs of decreased bandwidth utilization and increased computational complexity. While FEC for the base layer (BL) of an FGS video can be justified by the importance of the BL, FEC for the enhancement layer (EL) is questionable. The first reason is that the BL is usually encoded at (very) low bitrate. Even if we choose high channel-coding rate, the effect on the size of the final output bitstream is not very significant as clearly shown in Fig. 6.4 where a fairly heavy protection results an increase in size by 1/3. However, applying FEC to the EL can result in very high bitrate because the bitrate for the EL is normally high. The second reason is that the EL is designed to be partially deliverable and it is simply impractical to determine the to-be-sent EL rate for each frame and set up an optimal FEC configuration for it in real-time. Based on these considerations, we prefer encoding the EL with inherent error-resilience over providing FEC to the EL. We can also safely presume that datagrams passed to the decoder are error-free because erroneous packets will be detected and discarded by the underlying protocol stacks. From the applications' point-of-view, the only transmission error that is observed is packet loss. If coupling exists between different packets, packet loss will be disastrous. One packet loss will render many received packets useless because of a *Domino Effect*. This is the justification behind the design of our error-resilient FGS codec. In our proposed codec, each generated EL packet is independently decodable. Each correctly decoded packet contributes to the visual

quality improvement of the reconstructed frames. The fine granular scalability is achieved with this mechanism.

In §6.2, we have already discussed the effects of employing FEC in streaming of non-scalable MPEG-4 videos by using analytical models and simulations. In this section, we will focus on the streaming of an FGS video. Specifically, we will study the efficiency of our proposed FGS video streaming mechanism, *i.e.*, protecting the BL with FEC whilst protecting the EL with error-resilient source coding. The WavFiGS source coder proposed in §5.2 was used to generate source trace files to be used in the simulation. Two trace files were generated for each sequence. One trace file contains data for the error-resilient EL while the other contains data for the EL without error-resilience. The BL data for the two trace files are identical. The MPEG-4 FGS trace files were not used because: 1) the reference software [77] we used could not decode erroneous bitstreams and 2) we have not designed an error-resilient EL for the MPEG-4 FGS coding.

The setup of the simulation was similar to what we have described in §6.2.2 except that the EL was involved. For every run of simulation, identical frames from a sequence and error patterns are used for the two trace files to ensure that the performance is compared under the same condition.

The following excerpt of an OTcl script depicts the network settings in our simulation:

```
...
set tfile [new VideoTracefile]
$tfile filename foreman_cif_bl.meta
set fgsfile [new VideoFGSTracefile]
$fgsfile filename foreman_cif_el.meta
#Choose the seed for RNG, if not specified,
#it will be automatically generated
```

```
$tfile set seed_ 48225

#Set up the streaming server here
set server [new Application/Traffic/VideoTrace/FECFGS]
$server attach-agent $udp0
$server attach-tracefile $tfile
$server set frameRate_ 25
$server set enableFEC_ 1
$server set NumGOPs_ 4
$server FEC cfg-file fec.cfg
$server FEC config GOP
$server attach-fgs-tracefile $fgsfile
$server set totalBitrate_ 5.4Mb

#Set up the streaming client here
set client [new Application/VideoDecoder/FECFGS]
$client attach-agent $udp1
$client set enableFEC_ 1
$client set NumGOPs_ 4
$client set bitRate_ 400kb
$client set frameRate_ 25
$client set lastFrame_ 299
$client set fgsBitrate_ 5Mb
$client set fgsBufferedTime_ 2.0
$client output-file foreman_cif_recv.meta
...
```

After setting up a video server and a client as in §6.2.2, a BL trace file is openned

for access. The starting frame in this BL trace file is determined by the member *seed_* of *$tfile*. The corresponding EL trace file is then opened and synchronized to the starting frame in the BL. A streaming server is then instantiated as *$server*, whose parameters are set accordingly. The configuration of the FEC is read from a configuration file, *fec.cfg*. This configuration file contains the (n, k) pairs for all three frame types. In our simulation, the GoP-based FEC was chosen since we have already shown that it performs better under practical setup. The number of packet blocks is specified by *NumGOPs_*, which must keep consistent between the server and the client. The client is instantiated as *$client*, in which the bitrates for both the BL and the EL are specified. The bitrates then jointly determine receiver buffer size with specified bitrates. Note that only the bitrate for the EL is used since the receiver buffer size for the BL can be inferred from the (n, k) pairs. The successfully delivered EL data are saved in the file *foreman_cif_recv.meta*, which is then fed back into the WavFiGS coder to generate reconstructed frames and calculate PSNR values.

In the simulations, a 5-second clip was streamed out of the sequence "Foreman". Since independent packet loss events were assumed in the setup, the length of clip did not have an impact on the simulation results. The 5-second clip consisted of about 125 compressed frames, the first of which was an I-frame. The BL of the sequence was compressed at 400 Kbps. The EL was compressed progressively to finest bit-planes without explicit rate-control as in the BL coding. However, we chose three EL bitrates, namely, 1, 3, and 5 Mbps. The combined bitrates for both layers were 1.4, 3.4, and 5.4 Mbps. The streaming server was responsible to allocate the EL rate budget among the frames. Different values of network loss probability, p, were used to study the performance of error-resilient EL coding. For each value of p, a reasonable FEC configuration was chosen for it. The specific values are shown in Table 6.4. Note that the values following the frame types Table 6.4 indicate the the

Table 6.4: The FEC configurations under different network loss probabilities.

	Frame FEC			GoP FEC
p	I(9)	P(6)	B(3)	$(n,\ k)$
0.001	1	1	0	(55, 51)
0.05	1	1	0	(55, 51)
0.2	4	2	0	(61, 51)
0.4	8	4	1	(79, 51)

length of the longest frame of that type in the clip when $s_{pkt} = 1000$ bytes.

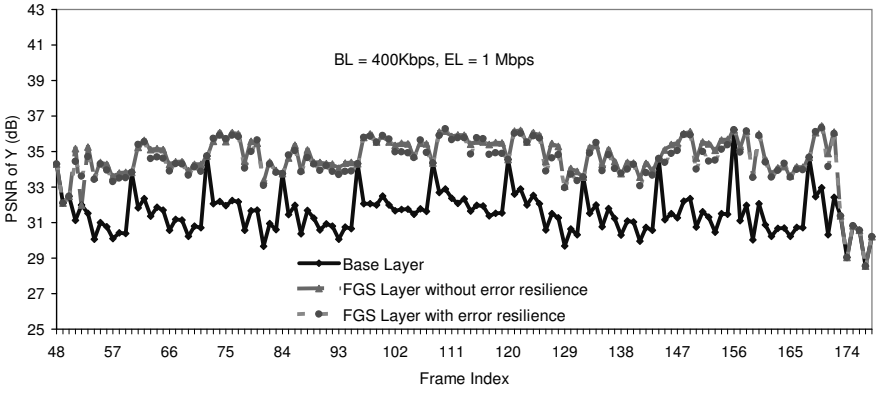
Fig. 6.7 illustrates the PSNR values of the reconstructed Y components when $p = 0.001$. It is shown that the results of the two reconstructed FGS sequences are barely distinguishable when we directly plot the PSNR values. To illustrate the difference better, we will only plot the PSNR gap between the two reconstructed FGS sequences.

$$\Delta PSNR(i) = PSNR_{NER}(i) - PSNR_{ER}(i) \tag{6.25}$$

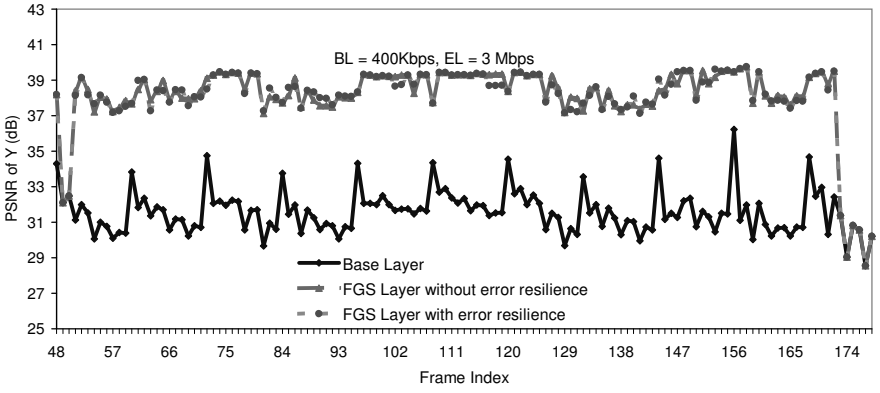
where $PSNR_{NER}(i)$ denotes the PSNR value of the reconstructed frame i employing the non-error-resilient EL and $PSNR_{ER}(i)$ denotes the PSNR value of the reconstructed frame i employing the error-resilient EL.

The PSNR gap plot corresponds to Fig. 6.7 is shown in Fig. 6.8.

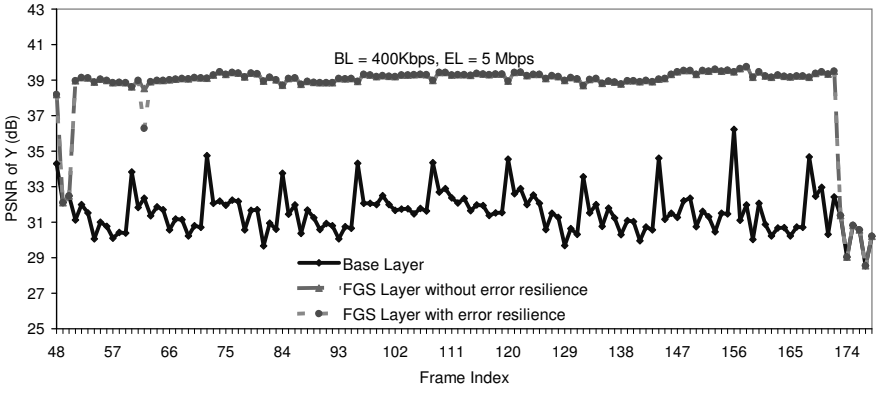
Fig. 6.8 reveals some interesting points regarding the FGS video streaming. In Fig. 6.8 (a), it is observed that in most reconstructed frames, the scheme without error-resilience achieves better PSNR results because of its better rate-distortion performance and small value of p. However, the negative spike in Fig. 6.8 (a) reveals the drawback of a non-error-resilient EL. The negative spike is caused by the loss of the highest bit-plane data, which renders the received data for lower bit-planes useless. In the tested sequence, the higher a bit-plane, the smaller its compressed size is because there are less significant coefficients at higher bit-planes. For example, the highest bit-plane of a slice often uses only 20-50 bytes. Therefore, all slices of a frame



(a)



(b)

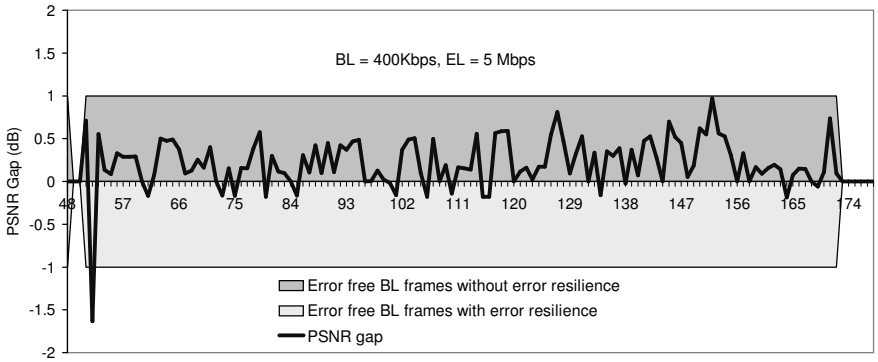


(c)

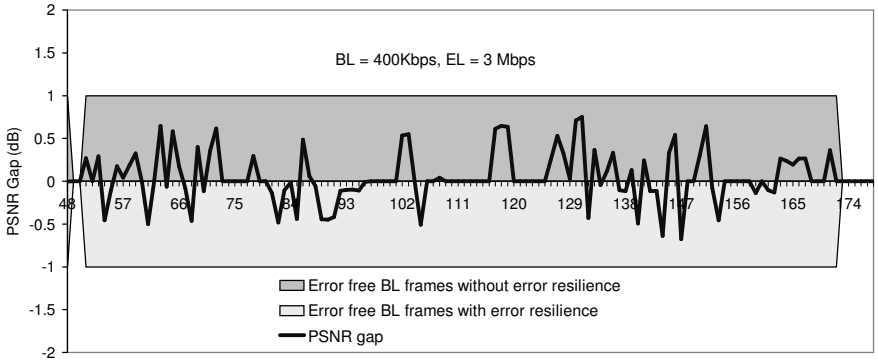
Figure 6.7: The PSNR of the reconstructed FGS frames at $p = 0.001$. The BL frames were protected with FEC and suffered no damage.

for the highest bit-plane can often be packed into the 1000-byte payload of a packet. If such a packet is lost, the total EL will be lost when we use the non-error-resilient scheme. Interestingly, such PSNR gap diminishes when the EL bitrate gets higher. This phenomenon can be explained by the fragmentation (packetization) strategy usually adopted in Internet video streaming [15]. Since each packet cannot contain information of more than one frame in a typical packetization scheme, it is inevitable that the payload size is variable. When the EL bitrate becomes higher, the size of the compressed lower bit-plane slice gets bigger and one packet may only contains data for one of such compressed bit-plane slice. For example, one non-error-resilient bit-plane slice occupies 670 bytes while its error-resilient counterpart may occupy 750 bytes. Although non-error-resilient scheme here saves 80 bytes in bit budget, this amount of saving cannot be translated into saving in packet budget because both will occupy a large portion of 1000-byte payload and the remaining space is not big enough to contain another full bit-plane slice. Hence, both schemes will end up reaching the identical PSNR results at a certain high bitrate. This point will be further discussed in the Appendix.

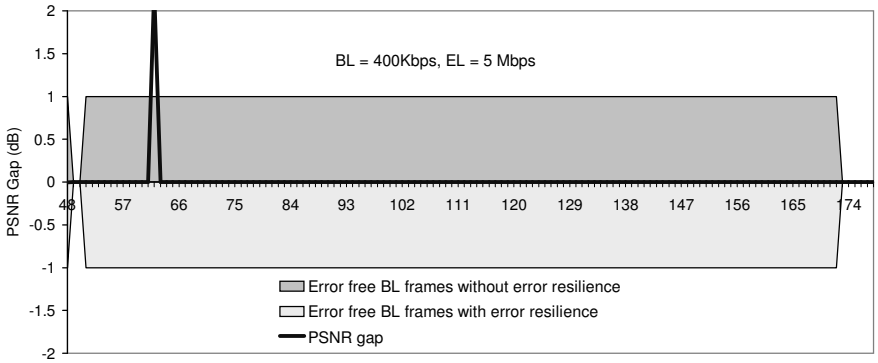
The results obtained under different streaming bitrates and network loss probabilities are plotted from Fig. 6.9 to Fig. 6.11. Note that in these figures, the PSNR values can only be compared when the BL frames are reconstructed as indicated by the boolean value 1 and -1. Here, -1 is treated as a boolean 1 with opposite polarity for plot clarity. When a boolean 0 is set for a frame m, it means that the BL of m cannot be decoded, the EL data of m are useless, and the $\Delta PSNR(m)$ cannot be calculated. The figures show that as the streaming bitrate gets higher and the network loss probability gets bigger, the proposed streaming strategy is more advantageous.



(a)

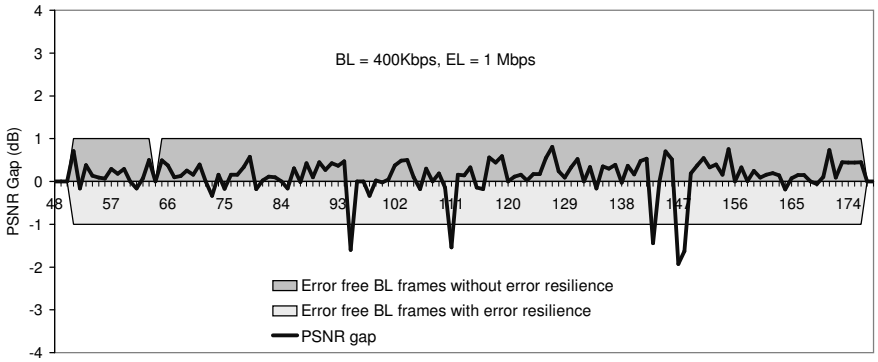


(b)

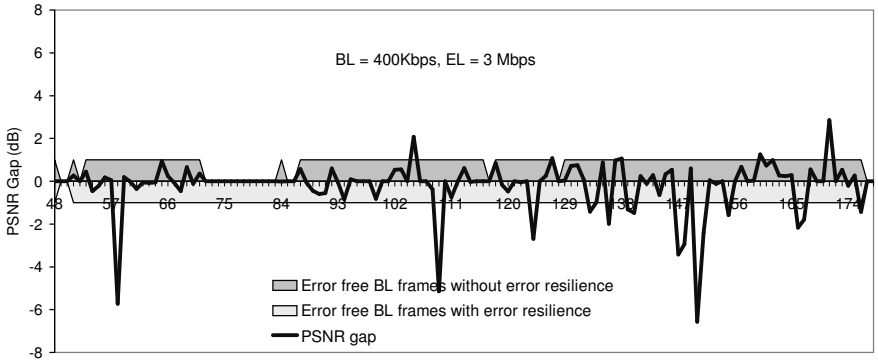


(c)

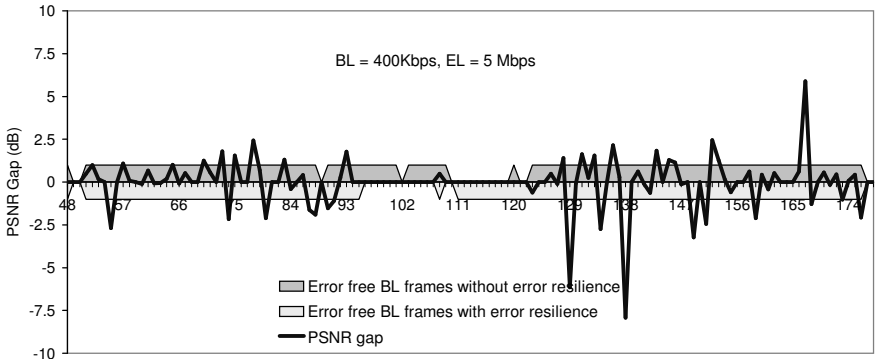
Figure 6.8: The PSNR difference of the two reconstructed FGS sequences at $p = 0.001$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity).



(a)

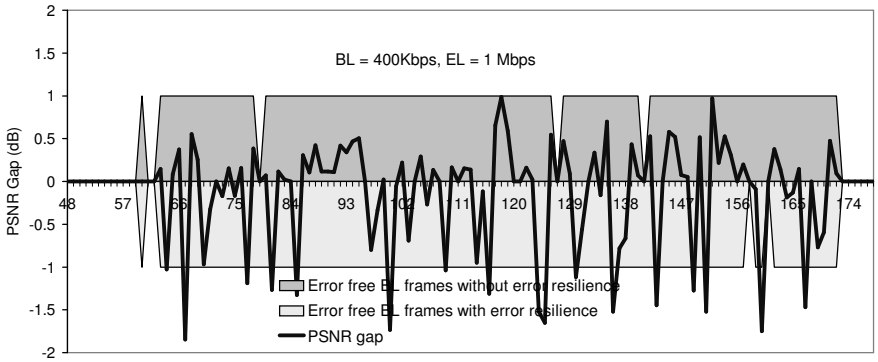


(b)

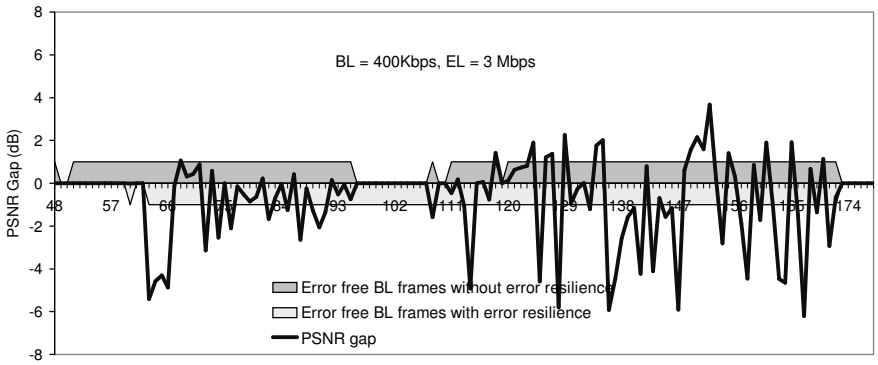


(c)

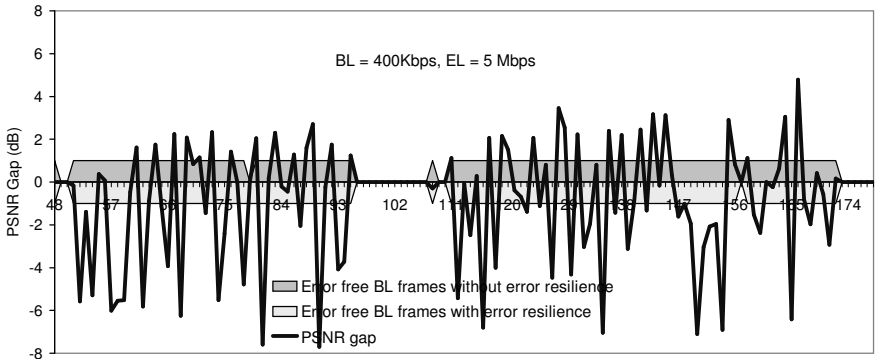
Figure 6.9: The PSNR difference of the two reconstructed FGS sequences at $p = 0.05$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity).



(a)

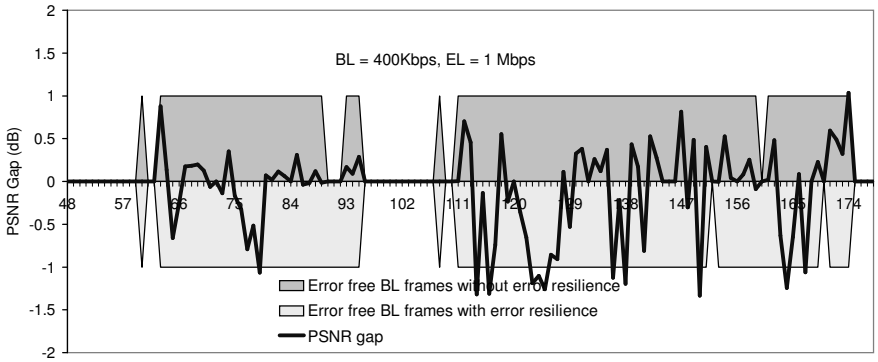


(b)

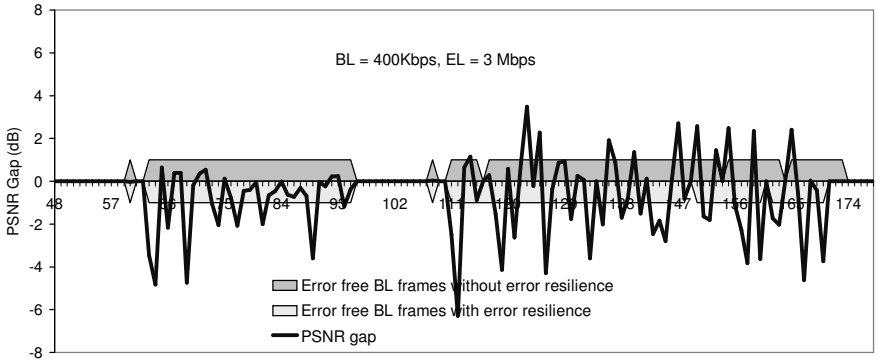


(c)

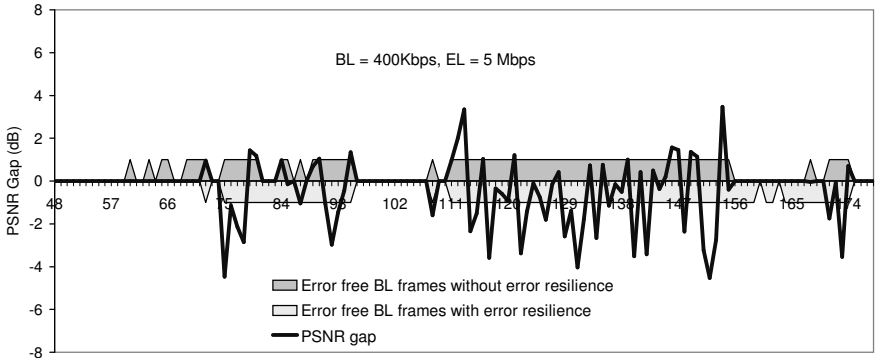
Figure 6.10: The PSNR difference of the two reconstructed FGS sequences at $p = 0.2$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity).



(a)



(b)



(c)

Figure 6.11: The PSNR difference of the two reconstructed FGS sequences at $p = 0.4$. The playable BL frames are indicated using boolean values (-1 is treated as boolean with different polarity).

6.4 Summary

We analyzed and simulated the transmission of compressed video over packet networks. We first showed mathematically that providing packet-level FEC protection to a whole group of compressed pictures should be preferred over providing different levels of such protection to different types of compressed pictures. The simulation we have conducted corroborates our mathematical analysis. In the latter part of this chapter, we demonstrated by simulation that the robust video coding scheme proposed in Chapter 5 indeed works better than schemes that does not incorporate error resilience.

Chapter 7

Conclusions

The major goal of this thesis work was to investigate the application of wavelets in error-resilient video compression systems and the performance of such systems in video streaming over packet networks. The major contributions of the research work are summarized in §7.1. Future research directions are identified in §7.2. The publications resulting from this research work are listed in §7.3.

7.1 Contributions

Internet video streaming is a key application in multimedia communication. Although some commercial products have been available since the early days of the Internet explosion, research on the transmission of compressed videos over connectionless networks is still in its infancy. Due to the bandwidth limitation and the statistical characteristics of the Internet traffic, the task of maintaining satisfactory quality in streamed video is challenging. Our research work has been focused on the design of a good wavelet FGS video coder using the JSCC approach. The following are the major contributions in this research work.

A digital video signal consists of a sequence of discretized still images. Any high performance video compression scheme starts with a good still image coder.

Wavelets, because of their excellent energy-compaction and multi-resolution capabilities, have been successfully applied in the compression of visual information, especially still image coding. Although the zerotree-based wavelet still image coding has become a well-established research area since it was proposed by J. M. Shapiro in 1993 [108], such zerotree-based schemes are not always suitable for all scenarios. The zerotree coding approach suffers from an entanglement of modeling, coding, and algorithmic issues, which makes it difficult to be implemented, especially in memory-constrained environments, such as portable devices.

In our research work, we have tried to build upon simple and efficient tools without sacrificing the compression quality in terms of PSNR. We have employed a low-complexity context modeling approach in embedded still image coding, which is based on the well-established principles of progressive bit-plane coding and RLC. Fairly complex entropy-coding methods, *e.g.*, adaptive AC, were not adopted. The outcome is promising and competitive against more complicated schemes, such as the SPIHT [105]. Similar context modeling approach has been extended to inter-component color image coding, with the proposed scheme outperforming the latest JPEG-2000 still image coding standard in terms of PSNR and color SNR. The source codes of the algorithms are ready for release.

The simplicity and efficiency of the proposed schemes has directly stimulated our research work in parallel image compression. We discovered that the information asymmetry between the encoder and the decoder could be exploited so that the compression of a bit-plane within a wavelet pyramid can be assigned to a PE (processor) as long as the whole pyramid is available to each and every PE. This is a breakthrough in wavelet parallel image coding since it provides a new dimension of processing flexibility in addition to the conventional parallelization approach, where images are divided into multiple blocks and each block is fed into one PE. This parallel bit-plane coding approach also eliminates the necessity for communication among

PEs because each bit-plane is encoded independently. It was also discovered that in the embedded image coding, the encoding order of the refinement bits within one bit-plane has little or no impact on the PSNR quality of the reconstructed image. This concept of parallel bit-plane coding is general enough that parallelization of other wavelet image coding schemes could also benefit from it. It is expected that this algorithm will be useful in image coding with large volumes of multiple processors. The algorithm has been thoroughly tested using multi-threading programming on a dual-processor workstation and the source codes can readily be released.

Although the embedded coding principle has been successful in wavelet image coding, we concluded that this principle is not suitable for wavelet video coding after an analysis on ME/MC techniques, quantization methods, and the statistics of motion compensated residual data. Instead, we believe that coding with MBs offers several advantages, such as error-resilience, adaptive-quantization, and compatibility with current coding standards. Based on the MB coding strategy, the proposed scheme achieves similar PSNR performance as the H.263 and is amongst the best wavelet video coders up-to-date.

To examine the proposed robust FGS video streaming strategy, *i.e.*, protecting the BL video with FEC and incorporating error-resilience into the EL video, we have designed a wavelet FGS video coder, WavFiGS, in which the BL coder is based on the MB coding strategy and the EL is based on the robust EG coding. The error-resilient EL of the WavFiGS was designed for packet-oriented environments where packet loss is considered the source of transmission errors. The EL bitstream can be packetized in such a way that each EL packet produced can be independently decoded and contribute to the improvement of the reconstruction quality. The rate-distortion performance of the WavFiGS is similar to that of the MPEG-4 FGS coder.

The streaming of the error-resilient wavelet FGS video has been analyzed using analytical models and simulations. FEC coding has been proposed for application

scenarios where feedback-based retransmission is either unavailable, as in satellite broadcasting, or impractical, as in multicast. An analytical video streaming model was proposed for optimally providing FEC in a hybrid video coding framework. The model takes into consideration the frame dependency within a GoP and compares different FEC mounting strategies. The analytical model can be used to optimally determine channel-coding rates according to packet loss probabilities. The validity of the model has been supported by simulation using real trace data. The simulation results clearly demonstrated that the proposed FGS streaming strategy is efficient and incurs only minor overhead.

To facilitate the study on the proposed streaming strategy, we have developed modules in C++ for the NS-2 network simulator. The modules simulate the functionalities of video streaming servers and clients. The modules were designed and implemented with generality in mind so that they can be used not only for the wavelet FGS video streaming but also for the streaming of videos encoded with other schemes, such as the MPEG-2 and the MPEG-4 FGS. The source codes can be readily compiled within the NS-2 environment and simulations can be conducted with composed OTcl scripts.

7.2 Future Work

The research work conducted so far made a breakthrough in parallel wavelet image compression and explored the frontiers of FGS video coding and streaming. More research work should be done to promote the adoption of parallel bit-plane coding in other computation-intensive image compression scenarios, to enhance the compression performance of the wavelet FGS video coder, and to improve video streaming quality based on the simulation platform we have constructed. We identify some future research areas as follows:

- Application of the parallel bit-plane coding:

  In this thesis, we have implemented the parallel bit-plane coding and tested it on a dual-processor personal workstation with a general operating system. The potential of this concept could be exhibited in at least two aspects: 1) Parallel bit-plane coding could readily be used in other embedded wavelet image algorithms, *e.g.*, PWC [69], with ease. Other zerotree-based schemes could also benefit from this concept during the parallelization process. This will prove the generality of the paralle bit-plane coding concept; and 2) Parallel bit-plane coding could be implemented in a real parallel processing platform and used for volume image processing.

- Enhancement of the WavFiGS coder:

  The compression performance of the coder can be improved in several ways. First, the BL coder could be implemented as a plug-in in the MPEG-4 encoder instead of the currently adopted method, *i.e.*, encoding I-frames with the PESD and encoding P- and B-frames with the MBSR in the MPEG-2 framework. Second, some rate-control technique can be applied in the EL encoding so that the performance can be improved in terms of PSNR. The compression quality of the EL can be further improved by studying the perceptual weighting of residual coefficients and the run-length statistics. Currently, encoding of all bit-planes in the EL starts with the same initial estimate of probability of run symbols. This is obviously not an optimal solution. Ideally, coding of bit-planes with the WavFiGS should like what is done in the MPEG-4 FGS coder, where different bit-planes are encoded with different VLC tables.

- Bandwidth-aware video streaming:

  In our video streaming simulations, we made the assumptions that network feedback is not available and the network loss probability does not change

during a streaming session. In practice, the network loss probability is a random variable and the network feedback is often available to a streaming server. Specifically, the RTP/RTCP protocol suite in the NS-2 can be implemented and be used in streaming simulations. Unfortunately, the RTCP protocol module in the NS-2 is not available yet. The streaming server can monitor the instantaneous bandwidth and dynamically allocate bandwidth among the BL, the EL, and the redundant information. These functionalities could be built upon our completed video streaming modules.

7.3 Publications

Part of the work presented in this thesis has been published in (or submitted to) the following journals and conferences:

- Y. Yuan and M. K. Mandal, "Novel embedded image coding algorithms based on wavelet difference reduction," accepted for publication in *IEE Proceedings Vision, Image & Signal Processing.*

- Y. Yuan and M. K. Mandal, "An embedded wavelet image coder with parallel encoding and sequential decoding of bit-planes," *Real-Time Imaging*, v. 10, n. 5, pp. 285-295, Oct. 2004.

- Y. Yuan and M. K. Mandal, "Context-modeled wavelet difference reduction coding based on fractional bit-plane partitioning," *IEICE Transactions on Information and Systems*, v. E87-D, n. 2, pp. 491-493, Feb. 2004.

- Y. Yuan and M. K. Mandal, "A spatial scalable wavelet video coder based on low-band-shifted hierarchical backward motion estimation and compensation," submitted to *Canadian Journal of Electrical and Computer Engineering.*

- Y. Yuan and M. K. Mandal, "Embedded color image compression with context-modeled wavelet difference reduction," *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, v. 3, pp. 61-64, May 2004.

- Y. Yuan and M. K. Mandal, "An embedded wavelet image coder with parallel encoding and sequential decoding of bit-planes," *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, v. 3, pp. 841-844, May 2004.

- Y. Yuan and M. K. Mandal, "Context-modeled wavelet difference reduction coding based on fractional bit-plane partitioning," *Proc. of IEEE International Conference on Image Processing (ICIP)*, v. 2, pp. 251-254, Sept. 2003.

- Y. Yuan and M. K. Mandal, "Low-band shifted hierarchical backward motion estimation and compensation for wavelet-based video coding," *Proc. of the 3rd Indian Conference on Computer Vision, Graphics and Image Processing*, pp. 185-190, Dec. 2002.

- Y. Yuan and M. K. Mandal, "Fast embedded image coding technique using wavelet difference reduction," *Proc. of SPIE: Electronic Imaging and Multimedia Technology III*, v. 4925, pp. 200-208, Oct. 2002.

- Y. Yuan and M. K. Mandal, "Backward motion estimation for low bitrate wavelet-based video coding," *Proc. of the International Symposium on Antenna Technology and Applied Electro-magnetics (ANTEM)*, pp. 36, July, 2002.

Bibliography

[1] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Trans. on Info. Theory*, 42(6):1737–1744, November 1996.

[2] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Trans. on Image Proc.*, 1(4):205–220, April 1992.

[3] R. Aravind, M. R. Civanlar, and A. R. Reibman. Packet loss resilience of MPEG-2 scalable video coding algorithm. *IEEE Trans. on Circuits Sys. Video Tech.*, 6(5):426–435, October 1996.

[4] S. Bajaj, L. Breslau, and S. Shenker. Uniform versus priority dropping for layered video. In *Proc. of ACM SIGCOMM Appl. Tech. Arch. Proto. for Computer Comm.*, pages 131–143, November 1998.

[5] A. Basu, I. Cheng, and Y. Yu. Multi-server optimal bandwidth monitoring for QoS based multimedia delivery. In *Proc. of IEEE ISCAS*, volume 2, pages 812–815, May 2003.

[6] V. Bhaskaran and K. Konstantinides. *Image and video compression standards: algorithms and architectures*. Kluwer Academic Publishers, Norwell, MA, 1997.

[7] J. Bolot and T. Turletti. Experience with rate control mechanisms for packet video in the Internet. *ACM SIGCOMM Computer Communication Review*, 28(1):4–15, January 1998.

[8] J. M. Boyce and R. D. Gaglianello. Packet loss effects on MPEG video sent over the public Internet. In *Proc. ACM Multimedia*, pages 181–190, 1998.

[9] R. W. Buccigrossi and E. Simoncelli. Image compression via joint statistical characterization in the wavelet domain. *IEEE Trans. Image Proc.*, 8(12):1688–1701, December 1999.

[10] C. S. Burrus, R. A. Gopinath, and H. Guo. *Introduction to wavelets and wavelet transforms: a primer*. Prentice Hall Inc., Upper Saddle River, NJ, 1998.

[11] K. Chan. China sets eyes on DVD. http://www.nikkeibp.com/nea/jan98/jannap/chinaside3.html.

[12] P.-Y. Cheng, J. Li, and C.-C. Jay Kuo. Multiscale video compression using wavelet transform and motion compensation. In *Proc. of Inter. Conf. Image Processing*, pages 606–609, 1995.

[13] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra. Error control for receiver-driven layered multicast of audio and video. *IEEE Trans. on Multimedia*, 3(1):108–122, March 2001.

[14] C. Chrysafis and A. Ortega. Efficient context based entropy coding for lossy wavelet image compression. In *Proc. IEEE DCC '97*, pages 241–250, 1997.

[15] M. R. Civanlar. *Compressed video over networks*, chapter 11, pages 433–464. Signal Processing and Communications. Marcel Dekker, New York, 2001.

[16] M. R. Civanlar, G. Cash, and B. Haskell. AT&T's error resilient video transmission technique. RFC 2448, November 1998, http://www.faqs.org/ftp/rfc/rfc2448.txt.

[17] D. Clark and D. Tennenhouse. Architecture considerations for a new generation of protocols. In *Proc. of ACM SIGCOMM '90*, pages 201–208, 1990.

[18] A. Cohen, I. Daubechies, and J. C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Math.*, 45(5):485–560, May 1992.

[19] T. M. Cover and J. A. Tomas. *Elements of information theory.* John Wiley & Sons, Inc., New York, 1991.

[20] C. D. Creusere. Image coding using parallel implementations of the embedded zerotree wavelet algorithm. In *Proc. SPIE Digital Video Compression: Algorithms and Technologies*, volume 2668, pages 82–92, 1996.

[21] C. D. Creusere. A new method of robust image compression based on the embedded zerotree wavelet algorithm. *IEEE Trans. on Image Proc.*, 6(10):1436–1442, October 1997.

[22] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[23] P. de Cuetos, M. Reisslein, and K. Ross. Evaluating the streaming of FGS-encoded video with rate-distortion traces. Technical Report RR-03-078, Institut Eurecom, June 2003.

[24] T. Dorsey. Streaming video entertainment opportunities, June 2001. http://www.streamingmedialand.com/VideoStreamingOpportunity.htm.

[25] N. Feamster and H. Balakrishnan. Packet loss recovery for streaming video. In *Proc. of the 12th Intl. Packet Video Workshop*, Pittsburgh PA, USA, April 2002. http://amp.ece.cmu.edu/packetvideo2002/papers/82-iccsmsmmst.pdf.

[26] F. H.P. Fitzek and M. Reisslein. MPEG–4 and H.263 video traces for network performance evaluation. *IEEE Network*, 15(6):40–54, November/December 2001.

[27] S. Floyd, H. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM*, pages 43–54, September 2000.

[28] J. E. Fowler. Qccpack: an open-source software library for quantization, compression, and coding. In *Proc. of IEEE DCC '00*, page 554, 2000.

[29] Jr. G. G. Langdon. An adaptive run-length coding algorithm. *IBM Technical Disclosure Bulletin*, 26(7B):3783–3785, December 1983.

[30] Jr. G. G. Langdon and J. Rissanen. Compression of black-white image with arithmetic coding. *IEEE Trans. on Comm.*, 29(6):858–867, June 1981.

[31] M. Ghanbari. Two-layer coding of video signals for VBR networks. *IEEE J. of Selected Areas in Comm.*, 7(6):801–806, June 1989.

[32] M. Ghanbari and V. Seferidis. Efficient H.261-based two-layer video codecs for ATM networks. *IEEE Trans. on Circuits Sys. Video Tech.*, 5(2):171–175, April 1995.

[33] S. W. Golomb. Run-length encodings. *IEEE Trans. on Info. Theory*, 12(3):399–401, July 1966.

[34] S. Graham, P. Kessler, and M. McKusick. Gprof: a call graph execution profiler. In *Proc. of the SIGPLAN Symposium on Compiler Construction*, volume 17, pages 120–126, 1982.

[35] R. W. Hamming. *Coding and information theory.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1980.

[36] M. Handley and V. Jacobson. SDP: session description protocol. RFC 2327, April 1998, http://www.faqs.org/ftp/rfc/rfc2327.txt.

[37] D. Hoffman, G. Fernando, V. Goyal, and M. R. Civanlar. RTP payload format for MPEG1/MPEG2 video. RFC 2250, Januray 1998, http://www.faqs.org/ftp/rfc/rfc2250.txt.

[38] Q. Hu and S. Panchanathan. Image/video spatial scalability in compressed domain. *IEEE Trans. on Ind. Electro.*, 45(1):23–31, February 1998.

[39] ISO/IEC JTC 1/SC 29/WG 1. ISO/IEC 15444-3: Information technology — JPEG 2000 image coding system: Part 3.motion jpeg 2000. September 2002.

[40] ISO/IEC JTC 1/SC 29/WG 1. ISO/IEC FCD 15444-1: Information technology — JPEG 2000 image coding system: Core coding system [WG 1 N 1646]. March 2000, http://www.jpeg.org/FCD15444-1.htm.

[41] ISO/IEC JTC 1/SC 29/WG 11. Generic coding of moving pictures and associated audio – part. 2: video, November 1994. ISO/IEC 13818-2.

[42] ISO/IEC JTC 1/SC 29/WG 11. Coding of audio-visual objects – part. 2: visual, amendment 4: streaming video profile, July 2000. ISO/IEC 14496-2/FPDAM4.

[43] ISO/IEC JTC1/SC29/WG11. Coding of audio-visual objects, Part-2 Visual. ISO/IEC 14496-2, December 1998.

[44] ISO/IEC-JTC1/SC29/WG11. Mpeg93/457: Mpeg video test model 5 (tm-5). April 1993.

[45] ITU-T. Video coding for low bit-rate communication. Recommendation H.263 Ver. 2, January 1998.

[46] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM '88*, pages 314–329, August 1988.

[47] A. K. Jain. *Fundamentals of digital image processing.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[48] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Trans. on Networking*, 11(4):537–549, August 2003.

[49] R. L. Joshi, V. J. Crump, and T. R. Fischer. Image subband coding using arithmetic coded trellis coded quantization. *IEEE Trans. on Circuits and Sys. for Video Tech.*, 5(6):515–523, December 1995.

[50] G. Karlsson and M. Vetterli. Packet video and its integration into the network architecture. *IEEE J. Selected Areas in Comm.*, 7(5):739–751, June 1989.

[51] J. H. Kasner, M. W. Marcellin, and B. R. Hunt. Universal trellis coded quantization. *IEEE Trans. Image Proc.*, 8(12):1677–1687, December 1999.

[52] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata. RTP payload format for MPEG-4 audio/visual streams. RFC 3016, November 2000, http://www.faqs.org/ftp/rfc/rfc3016.txt.

[53] B.-J. Kim and W. A. Pearlman. Low-delay embedded 3-D wavelet color video coding with SPIHT. In *Proc. SPIE Visual Comm. Image Proc.*, volume 3309, pages 955–964, 1998.

[54] B.-J. Kim, Z. Xiong, and W. A. Pearlman. Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT). *IEEE Trans. Circuits Sys. Video Tech.*, 10(12):1374–1387, December 2000.

[55] M. Krunz, R. Sass, and H. Hughes. Statistical characteristics and multiplexing of MPEG streams. In *Proc. of IEEE INFOCOM*, pages 455–462, 1995.

[56] R. Kutil. Approaches to zerotree image and video coding on mimd architectures. *Parallel Computing*, 28(7-8):1095–1109, August 2002.

[57] T.-W. A. Lee, S.-H. G. Chan, Q. Zhang, W.-W. Zhu, and Y.-Q. Zhang. Allocation of layer bandwidths and FECs for video multicast over wired and wireless networks. *IEEE Trans. on Circuits Sys. Video Tech.*, 12(12):1059–1070, December 2002.

[58] A. S. Lewis and G. Knowles. Image compression using the 2-D wavelet transform. *IEEE Trans. Image Proc.*, 1:244–250, April 1992.

[59] W. Li. Overview of fine granularity scalability in MPEG-4 video standard. *IEEE Trans. on Circuits and Sys. for Video Tech.*, 11(3):301–317, March 2001.

[60] K. K. Lin and R. M. Gray. Video residual coding using SPIHT and dependent optimization. In *Proc. of IEEE DCC '01*, pages 113–122, 2001.

[61] W.-K. Lin and N. Burgess. Listless zerotree coding for color images. In *Proc. 32nd Asilomar Conf. on Signals, Syst. and Computers*, volume 1, pages 231–235, 1998.

[62] D. Loguinov and H. Radha. Measurement study of low-bitrate Internet video streaming. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW)*, pages 314–329, November 2001.

[63] D. Loguinov and H. Radha. End-to-end rate-based congestion control: convergence properties and scalability analysis. *IEEE/ACM Trans. on Networking*, 11(4):564–577, August 2003.

[64] S. M. LoPresto, K. Ramchandran, and M. T. Orchard. Image coding based on mixture modeling of wavelet coefficients and a fast estimation-quantization framework. In *Proc. of IEEE DCC '97*, pages 221–230, 1997.

[65] J. Lu. Media streaming and broadcasting on the Internet, July 2000. http://www.streamingmedialand.com/StreamingOverview.pdf.

[66] J. Lu. Signal processing for Internet video streaming: a review. In *Proc. SPIE Image Video Comm. Proc.*, volume 3974, pages 246–259, April 2000.

[67] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. January 1997, http://www.psc.edu/networking/papers/tcp-friendly.html.

[68] S. G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Trans. PAMI*, 11(7):674–693, July 1989.

[69] H. S. Malvar. Fast progressive wavelet coding. In *Proc. IEEE DCC*, pages 336–343, 1999.

[70] M. K. Mandal, E. Chan, X. Wang, and S. Panchanathan. Multiresolution motion estimation techniques for video compression. *Opt. Eng.*, 35(1):128–136, January 1996.

[71] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek. Overview of JPEG-2000. In *Data Compression Conference Proceedings*, pages 523–541, 2000.

[72] D. Marpe and H. L. Cycon. Very low bit-rate video coding using wavelet-based techniques. *IEEE Trans. on Circuits Sys. for Video Tech.*, 9(1):85–94, April 1999.

[73] S. A. Martucci, I. Sodagar, T. Chiang, and Y.-Q. Zhang. A zerotree wavelet video coder. *IEEE Trans. Circuits Sys. Video Tech.*, 7(1):109–118, February 1997.

[74] K. Mayer-Patel, L. Le, and G. Carle. An MPEG performance model and its application to adaptive forward error correction. In *Proc. ACM Multimedia*, pages 1–10, December 2002.

[75] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM Computer Communication Review*, volume 26, pages 117–130, August 1996.

[76] N. Merhav and V. Bhaskaran. Fast algorithms for DCT-domain image down-sampling and for inverse motion compensation. *IEEE Trans. on Circuits Sys. Video Tech.*, 7(3):468–476, June 1997.

[77] Microsoft. ISO/IEC 14496 (MPEG-4) video reference software, July 2000. Ver. Microsoft-FPDAM1-1.0-000403.

[78] J. Mogul and S. Deering. Path MTU discovery. RFC 1191, November 1990, http://www.faqs.org/ftp/rfc/rfc1191.txt.

[79] G. Morrison and I. Parke. A spatially layered hierarchical approach to video coding. *Signal Proc.: Image Comm.*, 5(5-6):445–462, December 1993.

[80] MPEG Software Simulation Group. MPEG-2 Encoder/Decoder. July 1996, ftp://ftp.mpegtv.com/pub/mpeg/mssg/.

[81] A. N. Netravali and C. B. Rubinsein. Luminance adaptive coding of chrominance signals. *IEEE Trans. on Commun.*, 27(4):703–710, April 1979.

[82] J. Nonnenmacher, E. W. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. *IEEE Trans. on Networking*, 6(4):349–361, August 1998.

[83] A. Nosratinia and M. T. Orchard. A multi-resolution framework for backward motion compensation. In *Proc. of SPIE Electronic Imaging*, pages 190–200, 1995.

[84] The Dirac Team of BBC. The Dirac general purpose wavelet video coder. http://www.bbc.co.uk/rd/projects/dirac/, March 2004. Ver. 0.1.0.

[85] F. Ono, S. Kino, M. Yoshida, and T. Kimura. Bi-level image coding with MELCODE - comparison of block type and arithmetic type code. In *Proc. IEEE Globecom*, pages 255–260, November 1989.

[86] E. Ordentlich, M. J. Weinberger, and G. Seroussi. A low-complexity modeling approach for embedded coding of wavelet coefficients. In *Proc. of IEEE DCC '98*, pages 408–417, 1998.

[87] J.-A. Osipow. Smoothing the path of the broadband wagon. http://www.ipsos-ideas.com/articles/vol3-1.cfm.

[88] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: a simple model and its empirical validation. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 303–314, 1998.

[89] H. W. Park and H. S. Kim. Motion estimation using low-band-shift method for wavelet-based moving-picture coding. *IEEE Trans. Image Processing*, 9(4):577–587, April 2000.

[90] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Trans. on Networking*, 7(3):277–292, June 1999.

[91] V. Paxson and S. Floyd. Why we don't know how to simulate the internet. In *Proc. the 1997 Winter Simulation Conf.*, pages 1037–1044, 1997.

[92] W. B. Pennebaker and J. L. Mitchell. *JPEG still image data compression standard*. Van Nostrand Reinhold, New York, 1993.

[93] J. Postel. User datagram protocol. RFC 768, August 1980, http://www.faqs.org/ftp/rfc/rfc768.txt.

[94] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, November/December 2003.

[95] VINT Project. The Network Simulator - NS-2. http://www.isi.edu/nsnam/ns/.

[96] K. Ramchandran, A. Ortega, and K. M. Uz. Multi-resolution broadcast for digital HDTV using joint source channel coding. *IEEE J. of Selected Areas in Comm.*, 11(1):6–23, January 1993.

[97] D. W. Redmill and N. G. Kingsbury. The EREC: an error-resilient technique for coding variable-length blocks of data. *IEEE Trans. on Image Proc.*, 5(4):565–574, April 1996.

[98] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for Internet video streaming. *IEEE J. on Selected Areas in Comm.*, 18(12):2530–2542, December 2000.

[99] R. F. Rice. Some practical universal noiseless coding techniques. JPL Publication 79-22, NASA JPL, Pasadena, CA, March 1979.

[100] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Comm. Rev.*, 27(2):24–36, April 1999.

[101] L. Rizzo. pgmcc: a TCP-friendly single-rate multicast congestion control scheme. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 17–28, August 2000.

[102] L. Rizzo and L. Vicisano. RMDP: an FEC-based reliable multicast protocol for wireless environments. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(2):23–31, April 1998.

[103] J. Rogers and P. Cosman. Wavelet zerotree image compression with packetization. *IEEE Signal Proc. Letters*, 5(5):105–107, May 1998.

[104] J. Rosenberg and H. Schulzrinne. An RTP payload format for generic forward error correction. RFC 2733, December 1999, http://www.faqs.org/ftp/rfc/rfc2733.txt.

[105] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. on Circuits and Sys. for Video Tech.*, 6(3):243–250, June 1996.

[106] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. RFC 1889, January 1996, http://www.faqs.org/ftp/rfc/rfc1889.txt.

[107] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (RTSP). RFC 2326, April 1998, http://www.faqs.org/ftp/rfc/rfc2326.txt.

[108] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. on Acoustics, Speech and Signal Proc.*, 41(12):3445–3462, December 1993.

[109] K. Shen and E. J. Delp. Color image compression using an embedded rate scalable approach. In *Proc. of IEEE ICIP*, volume 3, pages 34–37, 1997.

[110] K. Shen and E. J. Delp. Wavelet based rate scalable video compression. *IEEE Trans. on Circuits Sys. for Video Tech.*, 9(1):109–122, February 1999.

[111] S. Shenker and J. Wroclawski. Network element service specification template. RFC 2216, September 1997, http://www.faqs.org/ftp/rfc/rfc2216.txt.

[112] J. W. Shwartz and R. C. Baker. Bit-plane encoding: a technique for source encoding. *IEEE Trans. Aerospace Electro. Syst.*, 2(7):385–392, July 1966.

[113] G. Strang and T. Nguyen. *Wavelets and filter banks.* Wellesley-Cambridge Press, Wellesley, MA, 1997.

[114] Y. Takashima, M. Wada, and H. Murakami. Reversible variable length codes. *IEEE Trans. on Comm.*, 43(2/3/4):158–162, Feb/Mar./Apr. 1995.

[115] T. K. Tan, K. K. Pang, and K. N. Ngan. A frequency scalable coding scheme employing pyramid and subband techniques. *IEEE Trans. on Circuits Sys. Video Tech.*, 4(2):203–207, April 1994.

[116] W.-T. Tan and A. Zakhor. Video multicast using layered FEC and scalable compression. *IEEE Trans. on Circuits Sys. Video Tech.*, 11(3):373–386, March 2001.

[117] D. Taubman. Kakadu Software, Ver. 4.0.3. http://www.kakadusoftware.com/.

[118] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Trans. on Image Proc.*, 9(7):1158–1170, July 2000.

[119] D. Taubman and A. Zakhor. Multi-rate 3-D subband coding of video. *IEEE Trans. Image Proc.*, 3(5):572–588, September 1994.

[120] Telenor. TMN H.263 Encoder/Decoder, June 1996. Ver. 2.0.

[121] J. Tian and Jr. R. O. Wells. Embedded image coding using wavelet difference reduction. In P. N. Topiwala, editor, *Wavelet image and video compression*, pages 289–302. Kluwer Academic, 1998.

[122] P. N. Topiwala, editor. *Wavelet image and video compression*. Kluwer Academic Publishers, Norwell, MA, 1998.

[123] M. J. Tsai, D. Villasenor, and F. Chen. Stack-run image coding. *IEEE Trans. Circuits Sys. Video Tech.*, 6(5):519–521, October 1996.

[124] T. Turletti and C. Huitema. RTP payload format for H.261 video streams. RFC 2032, October 1996, http://www.faqs.org/ftp/rfc/rfc2032.txt.

[125] ISI USC. Internet protocol. RFC 791, September 1981, http://www.faqs.org/ftp/rfc/rfc791.txt.

[126] ISI USC. Transmission control protocol. RFC 793, September 1981, http://www.faqs.org/ftp/rfc/rfc793.txt.

[127] P. Vaidyanathan. *Multirate systems and filter banks*. Prentice Hall Inc., Englewood Cliffs, NJ, 1993.

[128] V. A. Vaishampayan. Design of multiple description scalar quantizers. *IEEE Trans. on Info. Theory*, 39(5):821–824, May 1993.

[129] M. van der Schaar and H. Radha. Unequal packet loss resilience for fine-granular-scalability video. *IEEE Trans. on Multimedia*, 3(4):381–394, December 2001.

[130] M. van der Schaar and H. Radha. Adaptive motion-compensation fine-granular-scalability (AMC-FGS) for wireless video. *IEEE Trans. on Circuits Sys. Video Tech.*, 12(6):360–371, June 2002.

[131] J. Vass, B.-B. Chai, K. Palaniappan, and X. Zhuang. Significance-linked connected component analysis for very low bit-rate wavelet video coding. *IEEE Trans. on Circuits Sys. for Video Tech.*, 9(4):630–647, June 1999.

[132] J. S. Walker. Lossy image codec based on adpatively scanned wavelet difference reduction. *Opt. Eng.*, 39(7):1891–1897, July 2000.

[133] Y. Wang and Q.-F. Zhu. Error control and concealment for video communication: a review. *Proc. of the IEEE*, 86(5):974–997, May 1998.

[134] F. W. Wheeler and W. A. Pearlman. SPIHT image compression without lists. In *Proc. IEEE ICASSP*, volume 4, pages 2047–2050, 2000.

[135] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Comm. of ACM*, 30(6):520–540, June 1987.

[136] J. W. Woods and G. Lilienfield. A resolution and frame-rate scalable subband/wavelet video coder. *IEEE Trans. Image Proc.*, 11(9):1035–1044, September 2001.

[137] F. Wu, S. Li, and Y.-Q. Zhang. A framework for efficient progressive fine granularity scalable video coding. *IEEE Trans. on Circuits Sys. Video Tech.*, 11(3):332–344, March 2001.

[138] H. Wu, M. Claypool, and R. Kinicki. A model for MPEG with forward error correction and TCP-friendly bandwidth. In *Proc. of the ACM NOSSDAV*, pages 122–130, 2003.

[139] X. Wu. High-order context modeling and embedded conditional entropy coding of wavelet coefficients for image compression. In *Proc. 31st Asilomar Conf. on Signals, Sys., and Computers*, pages 1378–1382, 1997.

[140] Z. Xiong, K. Ramchandran, and M. T. Orchard. Space-frequency quantization for wavelet image coding. *IEEE Trans. on Image Proc.*, 6(5):677–693, May 1997.

[141] X. Yang and K. Ramchandran. Scalable wavelet video coding using aliasing-reduced hierarchical motion compensation. *IEEE Trans. Image Processing*, 9(5):778–791, May 2000.

[142] F. Zhai and T. N. Pappas. Motion-compensated wavelet video coding using adaptive mode selection. In *Proc. SPIE Visual Comm. Image Proc.*, volume 5308, pages 1362–1370, January 2004.

[143] Y.-Q. Zhang and S. Zafar. Motion-compensated wavelet transform coding for color video compression. *IEEE Trans. on Circuits Sys. for Video Tech.*, 2(3):285–296, September 1992.

[144] C. Zhu. RTP payload format for H.263 video streams. RFC 2190, September 1997, http://www.faqs.org/ftp/rfc/rfc1997.txt.

Appendix A

The Design and Implementation of The Streaming System

In this appendix, we discuss the design and implementation details of the simulated streaming system. The system has been implemented in the C++ language as modules for the NS-2 network simulator [95]. The streaming system design for non-scalable videos is presented in §A.1, in which streaming scenarios with and without FEC are both addressed. The implementation in §A.1 can be used for the BL streaming of an FGS video. The discussion in §A.2 is focused on the processing of the EL of an FGS video.

A.1 Streaming System for Non-Scalable Video

To study the performance of FEC in video streaming, the server and the client of a video streaming system are to be designed and implemented based on the underlying NS-2 platform. Since there is no public source of video streaming server and client available for the NS-2, we designed and implemented the behavioral models of video streaming servers and clients as the NS-2 modules. In our design, there are three modules in a complete streaming server/client system: a streaming server module

at the server side, a receiver buffer module at the client side, and a video decoder module at the client side. Note that video encoder module is omitted because we are only interested in streaming pre-stored videos here and source trace files are used.

A.1.1 Streaming Server Module

The streaming server here is responsible for accessing a source trace file, generating source datagrams, computing redundant datagrams, and scheduling the handover of a packet to the underlying UDP agent. An assumption here is that the FEC redundant datagrams are computed by the server. Otherwise, the redundant datagrams can be computed during the encoding process and pre-stored as well.

The manipulation of an MPEG-4 trace file is accomplished by an implemented C++ class *VTraceFile*. *VTraceFile* stores information such as the length of the trace in number of frames and the starting point within the trace file. The starting point is random so that concurrent streaming requests generated by multiple clients on one server can be simulated. Each read operation by *VTraceFile* inputs one line from the trace file and the line is stored in a data structure called *VTraceRecord*:

```
typedef struct VTraceRecord {
        int frame_no; // frame no. in encoding order
        char type; // frame type
        int time; // replay time
        int size; // frame size in bytes
};
```

The streaming server is designed as a C++ class *FECVideoTrace*, which is derived from the class *TrafficGenerator* provided in the NS-2. *FECVideoTrace* provides the server-side buffering for generation of source and redundant datagrams. The number of source datagrams generated for a frame is determined by *VTraceRecord.size*.

$$N = \left\lceil \frac{VTraceRecord.size}{s_{pkt}} \right\rceil \tag{A.1}$$

where $\lceil \cdot \rceil$ denotes a ceiling operation. In our simulated packetizer, the length of the first $N-1$ source datagrams is s_{pkt} bytes and the last packet has a length of $(VTraceRecord.size - (N-1)s_{pkt})$ bytes. Because the size of the compressed frames varies, the last source datagram for a frame is often under-loaded, *i.e.*, $(VTraceRecord.size - (N-1)s_{pkt}) \leqslant s_{pkt}$. It is possible to pack such packets full using data from the next frame. But such action is generally avoided since it contradicts the principle of the ALF [17]. As discussed in [15], the following fragmentation rules apply when bitstream is packetized:

1. The video sequence header, if always, should always be at the beginning of the payload of a packet.

2. The GoP header, if always, should always be at the beginning of the payload, or follow a video sequence header.

3. The picture header, if always, should always be at the beginning of the payload, or follow a GoP header.

These rules have been applied in the payload format standard for the MPEG-1/2 Internet streaming [37]. Since each picture header is at the beginning of the payload, or source datagram, it is easy to see that equation (A.1) still holds. The difference between our simulation and some specific payload format is that the size of individual packets varies while the total payload length in bytes and the generated number of packets are identical. This difference does not have an impact on the simulation results because of the adoption of the PFR as the evaluation metric. When the PFR is evaluated, any packet loss for a frame will render that frame unplayable. In other words, as long as the number of packets for each frame remains the same, any type of packetization scheme can be used in simulation and achieve an identical PFR.

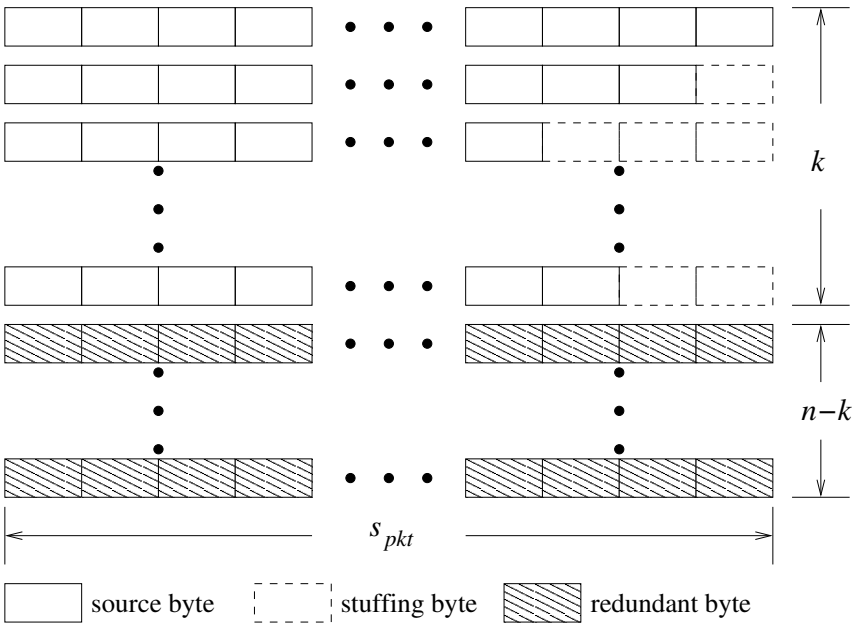


Figure A.1: The generation of FEC redundant packets.

In order to generate redundant datagrams, multiple frames must be read from the bitstream and fragmented into source datagrams first. Then the generator matrix **G** defined in equation (6.1) will be employed to generate redundant datagrams. This procedure is illustrated in Fig. A.1. Since some source datagrams are not exactly s_{pkt} bytes long, stuffing bytes are needed for them when redundant datagrams are generated. Bytes in a redundant datagram is generated sequentially. We use the first bytes of k source datagrams to generate the first bytes of $n - k$ redundant datagrams. Then we move on to the second bytes, and so on until it ends with the last bytes. Note that we need to regenerate **G** each time there is a change of value in an (n, k) pair. This regeneration system overhead is not negligible when we want to adaptively adjust the value(s) of an (n, k) pair to achieve dynamic FEC coding rates in implementing a streaming server.

There is a parameter file that stores the (n, k) pairs, which are read by *VTraceFile* before generating redundant datagrams. For Wu's frame-based FEC model [138], three (n, k) pairs are specified with each designated for one of the three frame types.

For our GoP-based FEC model, only one pair is needed for the whole GoP. Because of the system overhead associated with adjusting the value(s) of an (n, k) pair, caution must be advised on choosing the values of (n, k) so that

$$N_{t,m} \leqslant k_t,\ t \in \{I, P, B\} \tag{A.2}$$

where (n_I, k_I) denotes the (n, k) pair for I-frames, and so forth. $N_{I,m}$ denotes the number of source datagrams for the m-th I-frame during the streaming session, and so forth. For the proposed model, there is only one pair, (n_{gop}, k_{gop}), which satisfies

$$\sum_{i=1}^{(1+N_P)(1+N_{BP})} N_{i,m} \leqslant k_{gop} \tag{A.3}$$

where $N_{i,m}$ denotes the i-th frame in the m-th GoP. Although we may not want to change the pair values during a streaming session, it is generally impractical to use a fixed (n, k) pair throughout a long video streaming session. The reason is that scene changes tend to change the statistics of the frame size. As shown by an example in Fig. A.2, the size of the frames varies from hundreds of bytes to thousands of bytes. It should be more suitable to change the value(s) of the (n, k) pair according to the characteristics of a compressed video instead of using the same values unchanged throughout the whole streaming session.

After the generation of redundant datagrams, the transmission order of the packets is to be determined. It is natural to transmit the source datagrams before the redundant datagrams so that the source datagrams can be decoded right away when delivered. But scheduling the transmission of the redundant datagrams is a little more complicated. In the frame-based FEC, the redundant datagrams for a frame could be sent right after the source datagrams of the frame are sent. The redundant datagrams could also be sent sometime later leaving a temporal gap, t_g, between the transmission of source datagrams and redundant datagrams of the same frame.

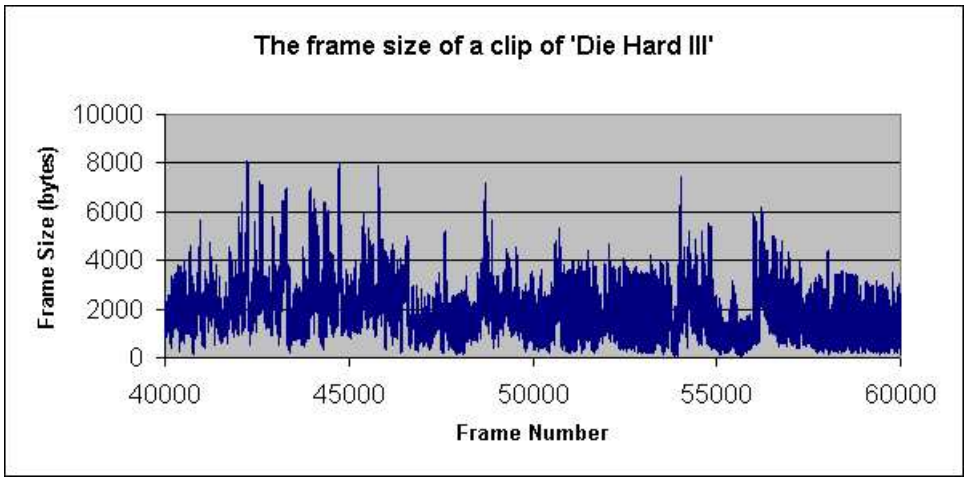


Figure A.2: An example on the frame size of compressed video.

This temporal gap should not cause any performance difference provided that packet loss events are independent, *i.e.*, binomial distribution fits perfectly. However, as suggested in [62, 90], the packet loss events are often bursty. Bursty losses reduce the effectiveness of the FEC when source and redundant datagrams are transmitted back-to-back. In our streaming server model, we use sender buffering to adjust the temporal gap so that bursty loss events can be dealt with. In both models we have discussed, the total number of transport packets for a GoP, N_{gop}, is fixed. In the frame-based FEC, N_{gop} can be calculated as

$$N_{gop} = n_I + N_P n_P + N_{BP}(1 + N_P)n_B \tag{A.4}$$

while in the GoP-based FEC, it can be calculated as

$$N_{gop} = n_{gop} \tag{A.5}$$

In our simulation model, the transmission order of these N_{gop} packets is shown in Fig. A.3. Here, transport packets representing B GoPs are grouped together and sent in an interleaved manner. From equation (6.9), it is clear that a GoP should be

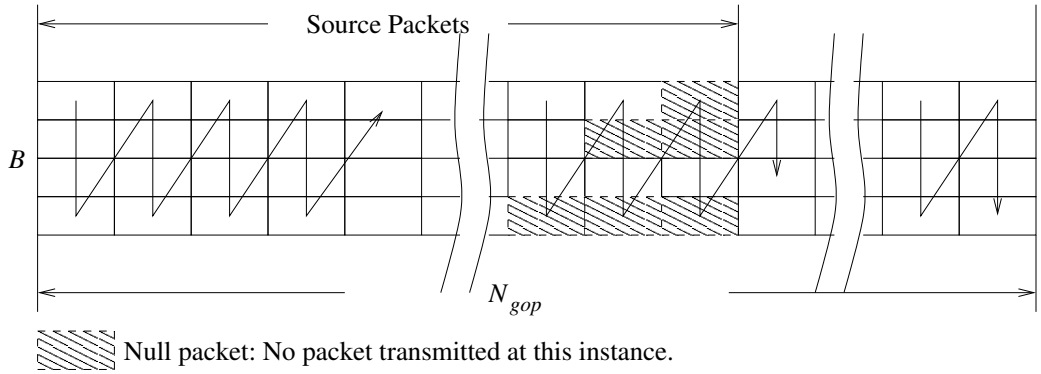


Figure A.3: Packet scheduling for the transmission of groups of packets.

completely sent in $1/G_s$ second in average. In our designed server module, a packet is sent per fixed interval ΔT:

$$\Delta T = \frac{1}{G_s \cdot N_{gop}} \tag{A.6}$$

Note that there exist some null packet locations in the transmission schedule. This is caused by the varying size of frames in packets. At null packet locations, no packet is transmitted. This means that near the end of source datagrams, the inter-packet interval is not exactly ΔT but several times of ΔT.

The parameter B is brought in to handle the bursty loss events. Assume $B = 1$ and the frame-based FEC is simulated, the temporal gap, t_g, ranges from $(N_{gop} - K_{gop}) \cdot \Delta T$ to $K_{gop} \cdot \Delta T$, where K_{gop} is the total number of source datagrams in a GoP.

$$K_{gop} = k_I + N_P k_P + (1 + N_P) N_{BP} k_B \tag{A.7}$$

The temporal gap can be dynamically adjusted to handle busty loss events by changing the value of B because now we have

$$B(N_{gop} - K_{gop})\Delta T \leqslant t_g \leqslant BK_{gop}\Delta T \tag{A.8}$$

where we assume that $N_{gop} < 2K_{gop}$. The larger the value of B, the better the streaming performance in presence of bursty loss events. The price paid for this performance gain is the size of the sender-side buffer, which linearly increases with B.

A.1.2 Receiver Buffer Module

If the packet delivery delay between the sender and the receiver is fixed, there would have been no need for a receiver buffer. However, this is not the case in practice. The delivery delay consists of two components: a *propagation delay* and a *queuing delay*. The propagation delay depends on the physical distance and is constant. The queuing delay on routers depends on the network congestion and may vary drastically.

The objective of using a receiver buffer is to reduce packet loss due to the delivery delay. The probability of delay loss is inverse-proportional to the size of receiver buffer. In an extreme situation, if a full video can be buffered before being replayed, there is no packet loss due to delay. In that case, the delay caused by buffering may be intolerable to viewers.

In our receiver buffer module, the concept of *buffer page* is used. Each page consists of B blocks of packets. Each block contains all transport packets for a GoP. Hence, the buffer delay caused by one buffer page is

$$d_{page} = \frac{B}{G_s} \tag{A.9}$$

Note that B is the same as defined in equation (A.8) and is not controllable by the receiver buffer. In fact, the size of the receiver buffer and/or the initial decoding delay is adjusted by changing the number of buffer pages. In order to start the decoder, at least one page of the receiver buffer must have been filled. Once the decoder starts, the buffer only accepts valid packets. The validity of a packet is

determined by the frame being decoded and the size of the buffer pages. Suppose the current buffer page starts from the k-th GoP, the number of the frame being decoded is f_{curr}, the buffer page size is p, and the packet belongs to frame $f_{i,m}$ of the m-th GoP. Then the packet is only valid when

$$\begin{cases} f_{i,m} & \geqslant & f_{curr} \\ m & \leqslant & k + pB - 1 \end{cases} \tag{A.10}$$

A.1.3 Video Decoder Module

A video decoder module retrieves packets from a receiver buffer periodically for decoding. In our module, the decoder accesses the buffer every $1/f$ second, where f is the frame rate of the video. A page-ready signal from the receiver buffer is used to trigger the decoding process. In our simulation, we always assume that decoding a frame takes less than $1/f$ second. The decoder simply counts the source datagrams of a frame delivered up to the time the buffer is accessed for that frame. If all source datagrams are either received or recovered from redundant datagrams, the frame is labelled *Good*. If lost packets for one frame is unrecoverable, it is labelled *Damaged*. If packets for one frame are either received or recoverable, but the frame are undecodable because its reference frame(s) is (are) not *Good*, the frame is labelled *Infected*. If both the packets are not recoverable and the reference frame(s) is (are) not *Good*, the frame is labelled *Both*.

All possible situations for a decoded frame are listed in table A.1, where X stands for *Not Applicable*, R stands for *Received* or *Recoverable*, and L stands for *Lost*.

The PFR can be calculated by counting the number of *Good* frames and the number of all frames streamed. However, if we like we could deduce more information from frames labelled otherwise.

Table A.1: Labels for decoded frames under all possible situations.

Frame Type	Packets	Reference Label	Label
I	R	X	G
I	L	X	D
P	R	G	G
P	R	D/I/B	I
P	L	G	D
P	L	D/I/B	B
B	R	(G,G)	G
B	R	other	I
B	L	(G,G)	D
B	L	other	B

A.2 Streaming System for FGS Video

Similar to the design in §A.1, a streaming server, a receiver buffer, and an FGS video decoder were designed as NS-2 modules to simulate FGS video streaming. Many functionalities in FGS video streaming are similar to those designed for the non-scalable video streaming. However, since the EL is involved in an FGS video streaming, additional considerations must be addressed in the design process.

A.2.1 FGS Video Streaming Server

Many functionalities of an FGS streaming server are similar to the server discussed in §A.1 except that additional considerations must be addressed.

In addition to the BL trace file, whose format is identical to that of an MPEG-4 trace file and handled by the *VTraceFile* class, the EL trace file is handled by a new class, *VFGSTraceFile*. *VFGSTraceFile* stores information such as the length of the trace in number of frames and the starting point within the trace file. The starting point within an EL trace file is synchronized with that of its corresponding BL trace file. Each read operation by *VFGSTraceFile* inputs one line from the trace file and the line is stored in a data structure called *VFGSRecord*:

```
typedef struct VFGSRecord {
        int frame_no;
        char type; // frame type
        unsigned char bit_plane; // the index of bit plane
        unsigned char slice; // the index of slice
        int size;
};
```

The streaming server is designed as a C++ class *FECFGS*, which is derived from the class *FECVideoTrace* designed in §A.1.1. *FECFGS* provides server-side buffering for generation of source and redundant datagrams. Since the BL data for each and every frame must be streamed out and the streaming of the EL data is not mandatory, we employed a straightforward rate-allocation scheme, *i.e.*, evenly allocate rate-budget for each frame. Hence, the number of source datagrams generated for a frame is determined by

$$N = \left\lfloor \frac{R_{BL} + R_{EL}}{f \cdot s_{pkt}} \right\rfloor \tag{A.11}$$

where R_{BL} denotes the BL bitrate, R_{EL} denotes the EL bitrate, and f denotes the source frame rate. The number of datagrams allocated to the EL of each frame is jointly determined by equation (A.1) and equation (A.11).

$$N_{EL} = \begin{cases} N - N_{BL}, & \text{if } N > N_{BL} \\ 0, & \text{else} \end{cases} \tag{A.12}$$

where N_{BL} is determined by equation (A.1).

Since the size of a typical compressed bit-plane slice is much smaller than s_{pkt}, we generally want to pack multiple bit-plane slices into one source datagram. Packing of bit-plane slices shall conform to the following guide lines:

- Slices from higher bit-planes should be packed earlier (with higher streaming priority) because they are more important in terms of rate-distortion performance.
- One slice should not be further fragmented so that each datagram can be decoded independently.
- One source datagram should contain as many bit-plane slices as possible to increase the system throughput.

The packing of the EL source datagrams is described in Algorithm 7. Note it is assumed in the algorithm that the bit-plane slices has been ordered according to their streaming priority.

Algorithm 7 The pseudo-code for packing bit-planes for EL streaming.

```
/*N_EL specifies the number of EL source datagrams*/
/*m is the current frame no.*/
bit_plane_slice_packer(N_EL,m) {
    counter := 0;
    while (N_EL ≠ 0) do {
        current_slice := read_in_one_bit-plane_slice(m);
        counter := counter + current_slice.size;
        if (counter > s_pkt) {
            complete_one_datagram(); N_EL := N_EL − 1;
            if (N_EL > 0) {
                pack_into_new_datagram(current_slice);
                counter := current_slice.size;
            }
            else break;
        }
        else pack_into_current_datagram(current_slice);
    }
}
```

In the simulation model, the transmission order of the BL packets is shown in Fig. A.3 and the transmission interval of the packets is specified in equation (A.6). Although the BL packets are interleaved during transmission, the EL packets are not

interleaved because the error resilient packets are more robust to bursty errors and the management of the EL receiver buffer can be simplified. All EL packets for a GoP need to be sent within $\frac{1}{G_s}$ second, where G_s is defined in equation (6.9). The EL packet scheduling used in our simulation is to send out all EL packets for a frame every $1/f$ second, where f is the source frame rate.

A.2.2 FGS Video Client

In an FGS video client, the receiver buffer consists of two parts, *i.e.*, the BL buffer and the EL buffer. The size of the BL buffer is determined by the number of buffer pages discussed in §A.1. The size of the EL buffer is determined by the EL bitrate and the buffering time given by a simulation setup script.

A buffer-manager class, *BufferManager*, has been designed for EL datagrams as an NS-2 module. *BufferManager* allocates buffer space according to the simulation setup script and manages the buffer during a streaming session. The allocated buffer space is organized like a ring as shown in Fig. A.4. In order to access the buffered datagrams, two pointers are used to indicate the area of valid datagrams. The pointers can only move clockwise. When a packet is received, the payload (datagram) of this packet is stored in the buffer and the head pointer is moved accordingly. Whenever a datagram is taken from the buffer, the tail pointer is moved accordingly. When the buffer is full and a new packet arrives, both pointers are moved, which indicates an overflow status and the oldest datagram is discarded.

Whenever a new packet arrives, *BufferManager* checks its frame number to make sure the packet does not belong to a frame that is earlier than the to-be-decoded frame. If such a packet has been delayed for a long time, it is discarded without entering the ring buffer. *BufferManager* also sorts all datagrams in the decoding order whenever a new datagram is inserted into the buffer.

The FGS decoder retrieves datagrams from the BL receiver buffer and the EL ring

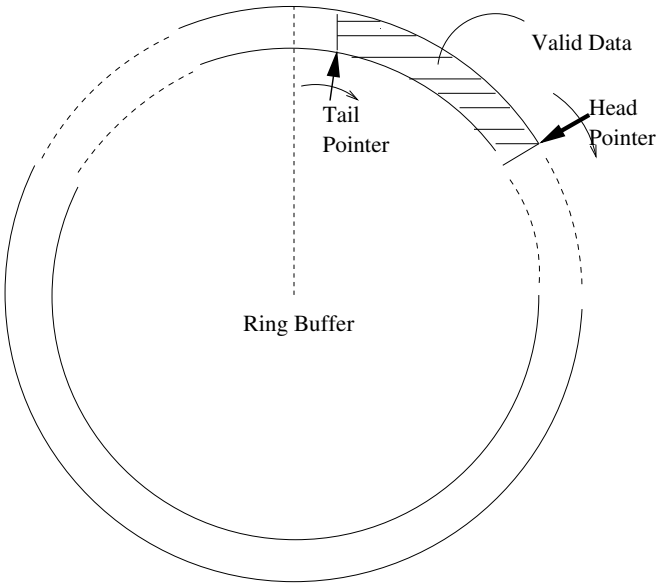


Figure A.4: The ring buffer used to store the delivered EL data. Both pointers can only move clockwise.

buffer periodically for decoding. The decoder accesses the buffers every $1/f$ second. A page-ready signal from the BL receiver buffer is used to trigger the decoding process. When the BL of a frame is successfully decoded, all EL datagrams belonging to it are taken out of the EL ring buffer and decoded. Otherwise, the EL datagrams are taken out and discarded.