

Portland State University
PDXScholar

Dissertations and Theses

Dissertations and Theses

2006

Efficient Support for Application-Specific Video Adaptation

Jie Huang

Portland State University

Let us know how access to this document benefits you.

Follow this and additional works at: http://pdxscholar.library.pdx.edu/open_access_etds

 Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Huang, Jie, "Efficient Support for Application-Specific Video Adaptation" (2006). *Dissertations and Theses*. Paper 2670.

[10.15760/etd.2664](https://doi.org/10.15760/etd.2664)

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.

DISSERTATION APPROVAL

The abstract and dissertation of Jie Huang for the Doctor of Philosophy in Computer Science were presented October 9, 2006, and accepted by the dissertation committee and the doctoral program.

COMMITTEE APPROVALS:

Wu-chi Feng, Chair

Andrew P. Black

Wu-chang Feng

Nirupama Bulusu

Fu Li
Representative of the Office of Graduate Studies

DOCTORAL PROGRAM APPROVAL:

Cynthia A. Brown, Director
Computer Science Ph.D. Program

ABSTRACT

An abstract of the dissertation of Jie Huang for the Doctor of Philosophy in Computer Science presented October 9, 2006.

Title: Efficient Support for Application-Specific Video Adaptation

As video applications become more diverse, video must be adapted in different ways to meet the requirements of different applications when there are insufficient resources. In this dissertation, we address two sorts of requirements that cannot be addressed by existing video adaptation technologies: (i) accommodating large variations in resolution and (ii) collecting video effectively in a multi-hop sensor network. In addition, we also address requirements for implementing video adaptation in a sensor network.

Accommodating large variation in resolution is required by the existence of display devices with widely disparate screen sizes. Existing resolution adaptation technologies usually aim at adapting video between two resolutions. We examine the limitations of these technologies that prevent them from supporting a large number of resolutions efficiently. We propose several hybrid schemes and study their performance. Among these hybrid schemes, *Bonneville*, a framework that combines

multiple encodings with limited scalability, can make good trade-offs when organizing compressed video to support a wide range of resolutions.

Video collection in a sensor network requires adapting video in a multi-hop store-and-forward network and with multiple video sources. This task cannot be supported effectively by existing adaptation technologies, which are designed for real-time streaming applications from a single source over IP-style end-to-end connections. We propose to adapt video in the network instead of at the network edge. We also propose a framework, *Steens*, to compose adaptation mechanisms on multiple nodes. We design two signaling protocols in *Steens* to coordinate multiple nodes. Our simulations show that in-network adaptation can use buffer space on intermediate nodes for adaptation and achieve better video quality than conventional network-edge adaptation. Our simulations also show that explicit collaboration among multiple nodes through signaling can improve video quality, waste less bandwidth, and maintain bandwidth-sharing fairness.

The implementation of video adaptation in a sensor network requires system support for programmability, retaskability, and high performance. We propose *Cascades*, a component-based framework, to provide the required support. A prototype implementation of *Steens* in this framework shows that the performance overhead is less than 5% compared to a hard-coded C implementation.

EFFICIENT SUPPORT FOR APPLICATION-SPECIFIC
VIDEO ADAPTATION

by

JIE HUANG

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
COMPUTER SCIENCE

Portland State University
2006

DEDICATION

To my parents, Zhao, and Jenny.

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Wu-chi Feng for his support and patience. He made research work less stressful by bringing in bigger challenges, such as playing foosball—for me, scoring in a foosball game is more difficult than getting a paper accepted by a top conference.

I would like to thank Professor Andrew Black and Professor Jonathan Walpole. I would not be here without their encouragement. I would also like to express my appreciation to Professor Wu-chang Feng, Professor Nirupama Bulusu, and Professor Fu Li for serving on my dissertation committee.

I have had the great pleasure to work with Buck Krasic, Kang Li, Ashvin Goel, Francis Chang, Jim Snow, Ed Kaiser, Chris Chambers, Phillip Sitbon, and Wilfried Jourve, although they always drove me crazy by talking about bicycles during lunch.

Finally, I would like to thank my husband Zhao for his constant support and my daughter Jenny for her passionate love. I would also like to thank my parents for their unconditional support.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
Chapter 1 Introduction	1
1.1 Motivation	2
1.2 Challenges	5
1.2.1 Challenges in Accommodating Large Variation in Resolution.....	6
1.2.2 Challenges in Supporting Video Collection	7
1.3 Thesis Statements	9
1.4 Dissertation Outline.....	10
Chapter 2 Background and Related Work	12
2.1 Video Compression	12
2.1.1 MPEG Overview	12
2.1.2 H.264 Overview	18
2.2 Video Adaptation.....	19
2.2.1 Tailoring Mechanisms	19
2.2.2 Adaptation Mechanisms	25
2.3 Sensor Networks and Video Sensors.....	26
2.3.1 Sensor Networks and Multi-hop routing	26
2.3.2 Video Sensors and Video-based Sensor Applications.....	27
Chapter 3 <i>Bonneville</i> : Supporting Wide-range Fine-grained Multi-resolution Video	
28	
3.1 Introduction	28
3.1.1 More Motivating Examples	28
3.1.2 Proposed Approach	30
3.2 Tailoring Mechanisms for Resolution Adaptation	32
3.2.1 A Single Encoding for All Resolutions	33
3.2.2 One Encoding Per Resolution	37
3.2.3 Hybrid Architectures and <i>Bonneville</i>	38
3.2.4 Mechanism Summary	39
3.3 Experiments and Analysis	39
3.3.1 Experimental Setup	40
3.3.2 Metrics	51
3.3.3 Experimental Results and Analysis	53
3.3.4 Supporting Multi-resolution Video	84
3.4 Conclusions	88

Chapter 4	<i>Steens</i> : Multi-hop Buffering and Adaptation for Video Collection in Sensor Networks	89
4.1	Introduction	89
4.2	Design of a Multi-hop Buffering and Adaptation Mechanism	92
4.2.1	Basic Tailoring Mechanism	93
4.2.2	Basic Adaptation Mechanism	93
4.2.3	Composition	94
4.3	Experimentation	100
4.3.1	Simulation Setup and Metrics	100
4.3.2	A Case for Hop-by-hop Adaptation	102
4.3.3	Exploring <i>Steens</i>	108
4.4	Conclusions	124
Chapter 5	<i>Cascades</i> : Supporting Video Adaptation in Sensor Networks	125
5.1	Introduction	125
5.2	The Cascades Architecture	127
5.3	Implementing Video Adaptation in Cascades	132
5.3.1	Programmability	133
5.3.2	Retasking	137
5.3.3	Performance Experiments	138
5.4	Conclusions	145
Chapter 6	Conclusions and Future Work	146
6.1	Research Contributions	146
6.2	Future Directions	148
6.2.1	Future Directions in Resolution Adaptation	148
6.2.2	Future Directions for Video-Based Sensor Applications	149
Reference		151

LIST OF TABLES

Table 2-1	A taxonomy of tailoring mechanisms for stored video.....	20
Table 3-1	Available display sizes.....	38
Table 3-2	Testing Sequences and Resolutions.....	41
Table 3-3	Compression efficiency of the first enhancement layer and the highest enhancement layer.....	58
Table 3-4	Compression efficiency of the highest resolution.....	65
Table 3-5	Bit-rates of derived resolutions.....	66
Table 3-6	Overall bandwidth Efficiencies for mechanisms in the one-encoding-for-all-resolutions architecture.....	67
Table 3-7	Comparison of compression efficiency for the highest resolution.....	73
Table 3-8	Overall bandwidth Efficiency for multiple scalable encodings.....	74
Table 3-9	Computational Cost for mechanisms in the one-encoding-for-all-resolutions architecture (<i>The Italian Job</i>).....	80
Table 3-10	Computational Cost for mechanisms in the one-encoding-for-all-resolutions architecture (<i>Street Corner</i>).....	80
Table 3-11	Computational Cost for hybrid architectures (<i>The Italian Job</i>).....	81
Table 3-12	Computational Cost for hybrid architectures (<i>Street Corner</i>).....	81
Table 3-13	Storage Cost for mechanisms in one-encoding-for-all-resolutions architecture.....	83
Table 3-14	Storage Cost for hybrid architectures.....	83
Table 5-1	JPEG performance.....	140
Table 5-2	JPEG IPP performance.....	141
Table 5-3	MPEG performance.....	142
Table 5-4	Code Sizes (JPEG-IPP).....	144
Table 5-5	Code Sizes (MPEG).....	144

LIST OF FIGURES

Figure 2-1	Intra-coding.....	13
Figure 2-2	YUV conversion and downsampling.....	14
Figure 2-3	A DCT and quantization example.....	15
Figure 2-4	Motion Estimation.....	16
Figure 2-5	Inter encoding.....	17
Figure 3-1	Encoding <i>The Italian Job</i> at different resolutions.....	29
Figure 3-2	The difference between lowering the spatial fidelity and downscaling the resolution.....	31
Figure 3-3	Generate DCT coefficients for low-resolution image in the DCT domain.....	35
Figure 3-4	Comparison of compression efficiency between re-encoding and transcoding by dropping AC coefficients.....	36
Figure 3-5	Multi-resolution mechanism summary.....	39
Figure 3-6	The H.264 encoder configuration file.....	43
Figure 3-7	The MPEG-2-like spatially scalable encoder.....	45
Figure 3-8	Dugad's spatially scalable encoder.....	46
Figure 3-9	Comparison of different representations for differential signals.....	48
Figure 3-10	Comparison of scaling algorithms.....	50
Figure 3-11	The resolution arrangement for mechanisms in the one-encoding-for-all-resolutions architecture.....	56
Figure 3-12	Bandwidth Efficiency of Different Algorithms for One-encoding-for-all-resolutions.....	57
Figure 3-13	The resolution arrangement for the combination of scalable encoding and DCT-coefficient dropping in the one-encoding-for-all-resolutions architecture.....	61
Figure 3-14	Comparison of the MPEG-2-like schemes with six layers and three layers combined with DCT-coefficient dropping.....	63
Figure 3-15	Comparison of the Dugad's scheme with six layers and three layers combined with DCT-coefficient dropping.....	64
Figure 3-16	The resolution arrangement for configurations in the hybrid architectures.....	69
Figure 3-17	Comparison of MPEG-2-like schemes in different configurations: one encoding or multiple encodings.....	71

Figure 3-18	Comparison of Dugad's schemes in different configurations: one encoding or multiple encodings.....	72
Figure 3-19	Comparison of re-encoding in different configurations: one encoding and multiple encodings.....	76
Figure 3-20	Compression efficiency of the combination of multi-encoding and DCT-coefficient dropping.....	77
Figure 3-21	Bandwidth efficiency for single resolutions.....	78
Figure 4-1	The basic adaptation mechanism and a simple prioritization mechanism.....	94
Figure 4-2	Examples of global prioritization.....	96
Figure 4-3	Signaling protocols for collaboration among sensor nodes.....	99
Figure 4-4	The network structures used in the simulations.....	101
Figure 4-5	The end-to-end reliability through hop-by-hop acknowledgement ...	103
Figure 4-6	Throughput and priority distribution for end-to-end adaptation and hop-by-hop adaptation.....	104
Figure 4-7	The frame rates.....	106
Figure 4-8	Wasted bandwidth.....	107
Figure 4-9	Throughput and priority distribution for three hop-by-hop adaptation systems.....	109
Figure 4-10	The frame rates (hop-by-hop adaptation).....	111
Figure 4-11	Wasted bandwidth.....	112
Figure 4-12	Throughput and priority distribution for <i>signaling 1</i> with different low watermarks.....	114
Figure 4-13	The frame rates (<i>signaling 1</i>).....	115
Figure 4-14	Wasted bandwidth.....	116
Figure 4-15	Throughput and priority distribution for <i>signaling 2</i> with different low watermarks.....	116
Figure 4-16	The frame rates (<i>signaling 2</i>).....	117
Figure 4-17	Wasted bandwidth.....	118
Figure 4-18	Fairness.....	120
Figure 4-19	Fairness.....	121
Figure 4-20	Throughput and priority distribution.....	122
Figure 4-21	Wasted Bandwidth (dropped data).....	123
Figure 5-1	An example of the overall architecture of <i>Cascades</i>	130
Figure 5-2	A simple adaptive video collection system.....	133
Figure 5-3	The construction of a video sensor capturing and adaptation system.	134

Figure 5-4	The capturing and adaptation script.....	134
Figure 5-5	Implementing signaling in Python.....	136
Figure 5-6	Retasking through dynamic reloading.....	137
Figure 5-7	The re-organized filter structure for retasking.....	138

CHAPTER 1

INTRODUCTION

The advent of digital video compression algorithms and standards [26][49] in the early 1990s has fostered the development of many video applications such as video conferencing and video on demand. For many of these applications, video adaptation is an indispensable tool to adjust their resource requirements to match the underlying resources supporting them. The goal of video adaptation, of course, is to adapt video to lower resource consumption while maximizing video quality. What makes this difficult is that the meaning of “quality” changes from one user to another, and from one application to another.

In this dissertation, we address how to adapt video to maximize video quality for different applications. Different applications have different resource constraints and different preferences on video quality. Therefore, they have different requirements on video adaptation technologies. As video applications are becoming more diverse, video adaptation must be specialized according to application requirements to maximize the video quality.

1.1 Motivation

Handling digital video can be burdensome for many computers or networks, especially as video resolutions continue to increase. For example, the H.261 video compression algorithms require approximately 968 million operations per second to compress CIF (358×288) resolution video at 30 frames per second (fps)[4]. This is with highly-optimized motion-estimation algorithms and fast DCT algorithms in place. Despite the large compression ratios of video compression algorithms, the data rate of a compressed video stream can still be several megabits per second (Mbps). Not only is the handling of digital video resource demanding, but the resource requirements are also bursty over time because of the temporal compression used between frames. Due to the high data rate and burstiness in resource requirements, it is often not feasible or cost-effective to provide resource guarantees for digital video across all resources.

Fortunately, many video applications can work without complete resource guarantees because they can tolerate some quality degradation. Video adaptation is the key to make these applications work when there are insufficient resources. It intelligently adapts video to lower resource consumption to meet resource constraints while providing the highest quality video possible (as defined by the user).

There are many ways to adapt a video stream; for example, either reducing the frame rate or downscaling the video resolution can reduce the bandwidth requirements needed to support it. The choice, however, is typically application-dependent. In the above example, frame rate reduction may work for video with little motion but not for

motion-intensive video. Conversely, resolution downscaling might be preferred by users with a palm-size display device but not by users watching the video on a larger PC display. Obviously, video adaptation needs the input from applications to maximize the video quality to a user's particular display and preferences.

Most existing video adaptation technologies are focused on providing continuous video for best-effort streaming applications such as video conferencing, webcasting, distance education, video surveillance, video on demand, and so on. The goal of these adaptation technologies is (i) to tailor video to fit available bandwidth and (ii) to deliver smooth video over bursty networks for uninterrupted playback. To tailor video to fit available bandwidth, the adaptation technologies use bit-rate reduction techniques that reduce the frame rate and/or lower the spatial fidelity. To deliver smoother quality video over bursty networks, they usually employ some form of buffering, in which larger buffers typically provide better video quality at the expense of latency. The buffer smoothes both network bandwidth fluctuations and the data rate fluctuations of compressed video.

As a variety of new video devices are emerging, video applications are becoming more diverse and video adaptation systems need to deal with more diversified application requirements. We will now describe two emerging application scenarios that have different requirements from existing applications and need support beyond existing adaptation technologies.

Scenario 1: Streaming high-resolution video to devices with widely disparate resolutions. In this scenario, a video server hosts a popular high-resolution video clip. The high-resolution video clip is generated either by an HD camcorder (1920×1080) or by stitching together video from multiple cameras (e.g. panoramic video) [8] [24]. To view this video stream, users can choose from a variety of devices covering a wide range of display sizes such as 96×64 pixels on a cell phone, 240×160 on a Palm device, 320×240 on a video iPod, 640×480 on an iPAQ, 1024×768 on a laptop, 1920×1080 on a HDTV, 2048×1536 on a PC monitor, or 2560×1600 on an Apple Cinema Display. To stream high-resolution video to a device with a small display, video adaptation technologies should downscale the video resolution to the display size because sending high-resolution video to the device is not as bandwidth-efficient as sending at the display resolution. Furthermore, it may cause significant processing problems on such devices. In this scenario, we are interested in the adaptation of video to a variety of display characteristics, where the range of display resolution variation may be greater than an order of magnitude.

Scenario 2: Collecting video in a sensor network. Oceanographers at Oregon State University would like to place a video camera every ¼ mile along the Oregon coast in order to observe near-shore phenomena [34]. This can be made possible by a class of new video capturing devices—video sensors. These video sensor nodes can capture, store, and process video and harvest energy from the environment for computation and networking. Furthermore, they can cooperate in order to pass data along the coast through other nodes to sink nodes with more power. This is a typical

application of a video sensor network, which collects video from multiple sensors and sends it through an ad hoc, multi-hop, store-and-forward network to a sink. While video collection does not have latency requirements as stringent as video streaming applications, it places four new requirements on adaptation technologies. First, it requires adaptation technologies to maximize video quality in an arbitrarily long time frame. Second, it requires adaptation technologies to work over multi-hop networks without end-to-end connectivity. Third, it requires adaptation technologies to ensure fair sharing of networking resources among multiple video sources, including both buffer and bandwidth resources. Finally, it requires adaptation technologies to adapt video based on video content or even to filter out unwanted video segments. For example, oceanographers might be interested in high quality video during high tide; in habitat monitoring, biologists might need only those video clips with a particular species in them.

These two scenarios show that while dealing with non-ideal network conditions is still a major responsibility of video adaptation, new application scenarios put new requirements on *how* it is accomplished. Existing technologies are still useful, but we need to further explore the adaptation space and to tailor adaptation technologies around these new application requirements.

1.2 Challenges

In this subsection, we discuss several challenges in meeting the requirements of accommodating large variations in resolution and supporting video collection in

video-based sensor networks. Our discussion is centered around two parts of a video streaming system. The *adaptation mechanism* is responsible for determining when and how much video to send across the network. It is also the mechanism that is responsible for determining the bandwidth for the video stream to match. The adaptation mechanism works in concert with the *tailoring mechanism*. The tailoring mechanism works in one of two ways. It either provides a video stream to the adaptation mechanism that is formatted in such a way that the adaptation mechanism can adapt the stream through dropping of marked data, or the tailoring mechanism reformats the video stream to a target rate based upon feedback received from the adaptation mechanism. A more detailed description of these mechanisms will be provided in Section 2.2.

1.2.1 Challenges in Accommodating Large Variation in Resolution

To display high-resolution video on small display devices, the video needs to be cropped to a smaller size and/or downsampled to a smaller resolution. Usually video is compressed; making changes to compressed video requires tailoring mechanisms such as re-encoding or transcoding. *Re-encoding* techniques decompress the stream into the pixel domain and then encode it again with new parameters. *Transcoding* partially decompresses the stream, manipulates the stream in the compressed domain in a way that approximates operations in the pixel domain, and re-encodes it. A summary of existing tailoring mechanisms will be presented in Chapter 2.

Accommodating large variation in resolution requires the tailoring mechanisms to support fine-grained region-of-interest (ROI) adaptation and resolution adaptation

over a wide range. Providing a large number of sub-regions and resolutions is challenging because the number of resolutions and the range of resolutions make tailoring difficult. For resolution adaptation, full re-encoding may be feasible for generating one new resolution, but encoding many resolutions at the same time is impractical even for modern computers. Fast transcoding can reduce the resolution in the compressed domain by up to a factor of eight but cannot downscale beyond that. We will describe the limitation of transcoding in more detail in Chapter 3.

In this work, we focus on supporting fine-grained resolution adaptation over a wide range of resolutions. We study how to organize and represent high-resolution video so it can be tailored to multiple resolutions efficiently.

1.2.2 Challenges in Supporting Video Collection

With the tailoring mechanism, we can tailor video to many different resolutions or target bit-rates. Still, we need an adaptation mechanism to determine the target bit-rates and a sending schedule so the resources can be utilized efficiently while maximizing the video quality. It is more challenging, though, to make the right decisions for video collection in a sensor network than in an IP-style network because there is no end-to-end connectivity in a sensor network and there are multiple video sources in a video collection application.

In an IP-style network, there is an end-to-end connection between a sender and a receiver, and the adaptation mechanisms make adaptation decisions according to the conditions of that connection. In a sensor network, the route from a video source to

the sink usually consists of multiple store-and-forward hops. This results in adaptation decisions that tend to be made based-upon the first hop. However, this is insufficient because the tailored video stream may not fit into the network bandwidth closer to the sink. The adaptation mechanisms need to consider the network conditions on all hops from the source to the sink without requiring end-to-end coordination. In addition, adaptation mechanisms also need to consider other video sources and not consume more than their fair share of resources. Making adaptation decisions based upon information from many sensor nodes that are not directly connected is a challenge.

Another challenge is implementing video adaptation within a sensor network with high performance and at the same time with sufficient flexibility to cope with the dynamic application requirements. High performance is important for video sensor applications because of the constrained resources in sensor network platforms and the large resource requirement for handling video. Meanwhile, flexibility is required by most sensor applications because application requirements often change after obtaining initial results. In addition, a sensor network is usually a distributed, embedded, and heterogeneous system; providing high performance and flexibility in such a system is always a challenge.

In this work, we study how to construct adaptation mechanisms that can collect the most useful video in a multi-hop store-and-forward network with multiple video sources. We also study the requirements of implementing video adaptation in a video sensor network and look for a framework to support the implementation.

1.3 Thesis Statements

In this dissertation, we address three video adaptation problems for emerging video application scenarios that are becoming feasible with the advent of new video capturing and display technologies.

Problem 1: What is the right tailoring mechanism to efficiently support fine-grained resolution adaptation over a wide-range of resolutions?

Thesis Statement 1: A combination of multi-encoding and scalable encoding/transcoding is necessary to tailor a video stream to multiple resolutions efficiently.

Problem 2: How can an adaptation mechanism efficiently collect video through a multi-hop store-and-forward sensor network to maximize the utility of video collected at the sink and minimize bandwidth wastage?

Thesis Statement 2: In-network adaptation and collaboration among store-and-forward sensor nodes are necessary to maximize the utility of video collected at the sink while minimizing wasted bandwidth.

Problem 3: How can we provide programmability, retaskability, and high performance for the implementation of video adaptation in sensor networks?

Thesis Statement 3: A component-based framework can make it easy to implement and retask video adaptation in sensor networks while retaining high performance.

1.4 Dissertation Outline

The rest of this dissertation is organized as follows.

Chapter 2 presents the necessary background for our work. Because our work is on compressed video and the basic concepts of video compression are important to understand it, we start with a short tutorial of video compression, using MPEG compression as an example. We also briefly review the more recent H.264 video compression standard. We then survey existing video adaptation technologies and sensor networking technologies.

Chapter 3 deals with the problem of how to organize and represent high-resolution video so it can be tailored to multiple resolutions efficiently. We first describe existing transcoding and scalable encoding algorithms that support multi-resolution video. We then present *Bonneville*, a hybrid architecture to support fine-grained video adaptation over a wide range of resolutions. Finally, we discuss the experimental setup and results.

In Chapter 4, we focus on video collection in ad hoc multi-hop store-and-forward sensor networks. We propose *Steens*, a multi-hop buffering and priority-based adaptation mechanism, for collecting video in a sensor network. We present the three components of *Steens*: prioritization, buffer management, and coordinating protocols. Finally, we demonstrate the advantages of *Steens* over traditional adaptation mechanisms through trace-driven simulations.

Chapter 5 discusses the problem of implementing video adaptation, especially content-based adaptation and adaptive collection, in a sensor network. First, we describe the requirements of in-network processing. We then discuss implementation technologies in scalar sensor networks and why they cannot be applied to video processing. We present *Cascades*, a component-based framework based on the scripting language Python to ease the implementation of video adaptation, and how it meets those requirements of in-network video processing.

In Chapter 6, we review the contributions of this dissertation and summarize key findings. Finally, we present future directions for research in this area.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we present the necessary background for this dissertation. We will describe (i) basic concepts in video compression, (ii) existing video adaptation technologies and their limitations, and (iii) recent developments in sensor networks and video sensors.

2.1 Video Compression

To help understand the motivation and details of our work, we present an overview of Discrete Cosine Transform (DCT) based video compression technologies. DCT-based compression has become extremely popular for imaging and video because of its high fidelity image reconstruction with high compression ratios [77]. We will use MPEG video as an example because it is the most commonly used video compression standard. We will also describe some of the new features in H.264, another popular DCT-based compression algorithm.

2.1.1 MPEG Overview

In this short introduction, we focus on the aspects of MPEG compression necessary to understand this work. A more detailed introduction to MPEG video is given by Gall [26].

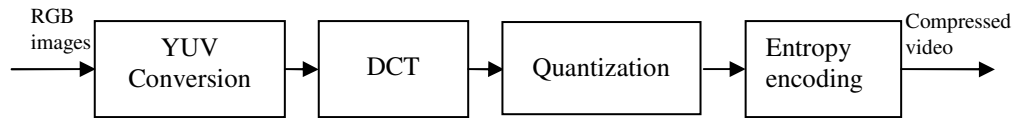


Figure 2-1 Intra-coding. This figure shows the four main steps involved in compressing a video frame. 1) Conversion of RGB color space to YUV color space, 2) Transformation into frequency domain via discrete cosine transform (DCT), 3) Quantization of DCT coefficients, and 4) Entropy encoding: Run Length Encoding (RLE) and Variable Length Coding (VLC).

There are two ways to encode individual frames in MPEG: intra-coding and inter-coding. Intra-coded frames are similar to JPEG images; they are encoded independently of other frames. In contrast, inter-coded frames are encoded by exploiting temporal redundancy among nearby frames.

The major steps for intra-coding are shown in Figure 2-1.

To prepare for compression, each image is divided into macroblocks of 16×16 pixels. In each macroblock, a conversion from the red, green, blue (RGB) color space into the luminance, chrominance, chrominance (YUV) color space is performed. This transformation allows the more important luminance component (Y) to be separated from the two chrominance channels (U and V). Since human eyes are less sensitive to chrominance channels, each 16×16-pixel chrominance block is typically sub-sampled into an 8×8 block whereas the luminance component is divided into four 8×8 blocks. This process is shown in Figure 2-2.

Next, the six 8×8 blocks are transformed into the frequency domain using discrete cosine transform (DCT). This transformation moves the lower frequency components

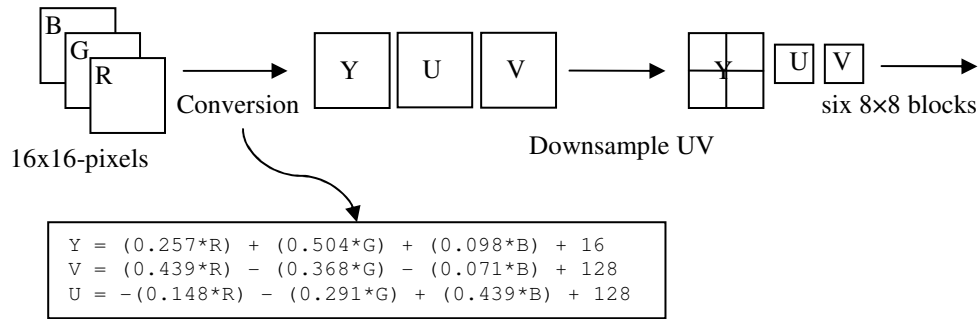


Figure 2-2 YUV conversion and downsampling

into the upper left corner of the block while moving the higher frequency components into the lower right corner. Thus, the average or the DC level of each block is in the upper left corner. The other 63 coefficients are called the AC coefficients. These 64 DCT coefficients are the values manipulated by many transcoding algorithms to alter the compressed video quality and/or the compressed stream size. In the quantization phase, the coefficients are quantized into discrete levels, typically giving coarser distinctions for higher frequency components. This is considered “lossy” and the quantization step size directly influences the compression ratio and the compressed video quality. Figure 2-3 intuitively shows how the combination of DCT and quantization reduces the number of coefficients to be encoded. There are many zeros at the lower right corner after quantization because the DCT coefficients are small and the quantization steps are large at the lower right corner.

Finally, the run-length encoded (RLE) coefficients for each block are compressed with variable length coding (VLC), a variant of Huffman encoding.

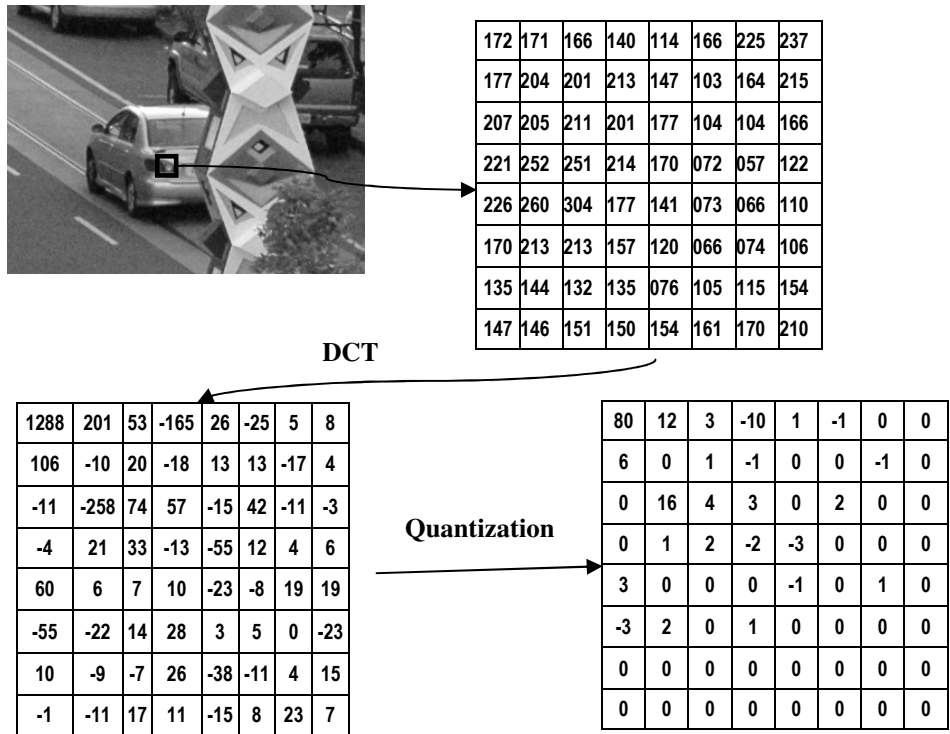


Figure 2-3 A DCT and quantization example

For inter-coded frames (sometimes referred as predictive-coded frames), there is an additional motion estimation (ME) step between the YUV conversion and the DCT. Motion estimation predicts a macroblock of pixel values using a motion-compensated macroblock from a reference frame. The location difference between the two macroblocks is called the *motion vector*; and the difference between the two macroblocks is called the *prediction error*. If the two macroblocks are similar enough and the prediction error for a block requires less bytes than the original block, the motion vector and the prediction error are encoded instead of the original block.

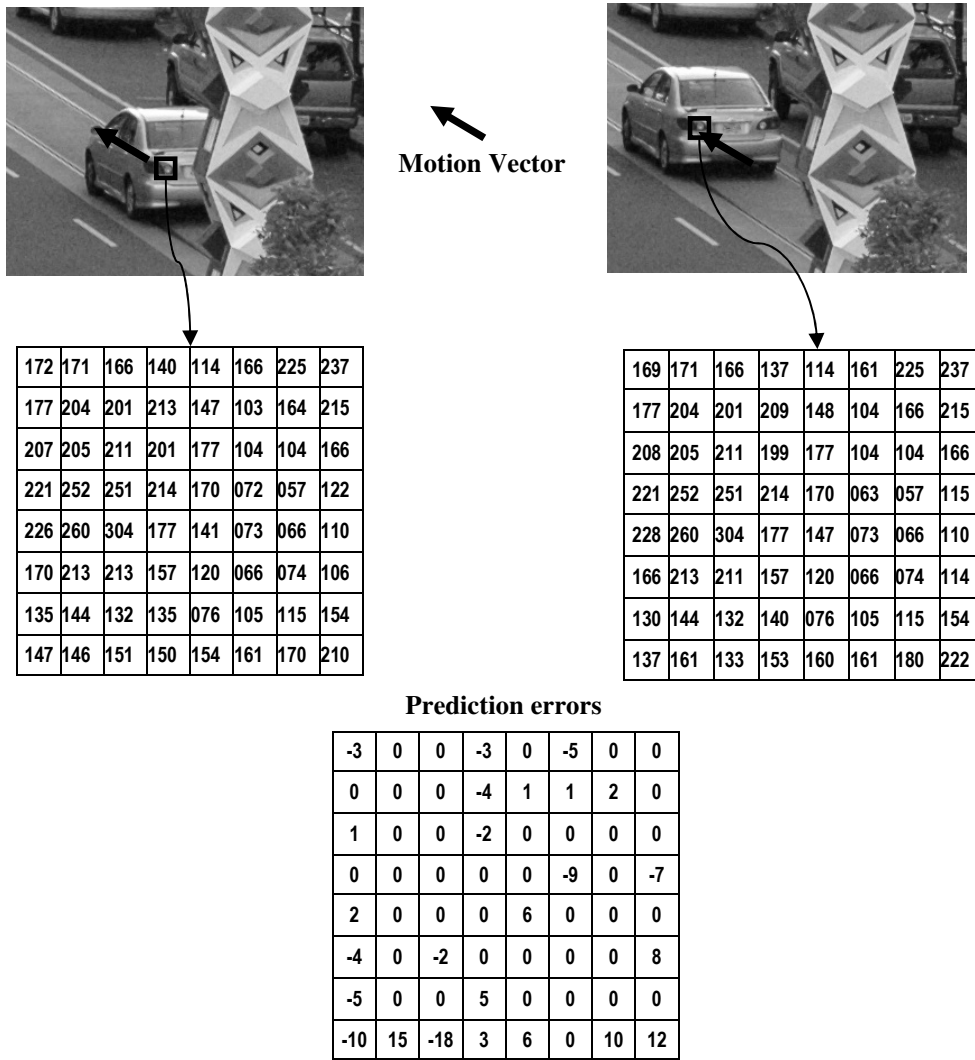


Figure 2-4 Motion Estimation

Figure 2-4 shows an example of a good match for motion estimation. Two consecutive frames are shown. The block being encoded is part of the left tail light of the car, shown in a black square in the right frame. The left tail light in the left frame is used as a reference, with the block being referenced shown in another black square. The Y components for both blocks and their differences are displayed below. The difference is so small that it can be skipped and only the motion vector is needed to encode the original block. That is, upon decompression, the decompressor simply

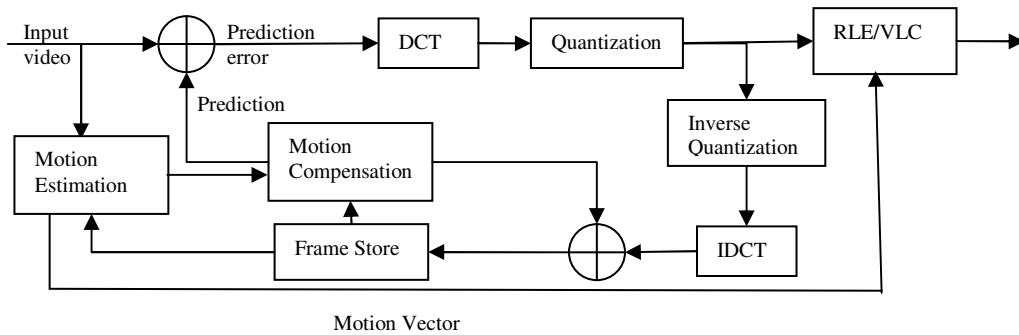


Figure 2-5 Inter encoding

needs to take a part of the reference frame (the reference block) and move it. Motion estimation is very important to video compression, but the process of finding a good match with the least prediction error can be very computationally demanding. A decoder (inverse quantization plus Inverse DCT) is included in the encoder, as shown in Figure 2-5, reflecting the actual reference frame the decoder will use.

Inter-coding introduces dependencies among compressed frames. For MPEG, the dependency relationship classifies compressed frames into three types: I frames, P frames, and B frames. I frames in MPEG are those frames within which all macroblocks are intra-encoded; their decoding does not depend on other frames. P frames and B frames have inter-encoded macroblocks; consequently they cannot be decoded before their reference frames have been decoded. There are two major differences between P frames and B frames. First, P frames are forward-predictive-coded with respect to a past frame while B-frames are bidirectionally-predictive-coded, requiring a preceding and a following frame. Second, a P frame can be a reference frame, for another P frame or a B frame, while a B frame cannot. When frame

dropping is used for video adaptation, the dependency relationship defines a partial order for drift-free dropping, that is, a reference frame should not be dropped before the frames that depend on it.

Most major block-based video compression standards, including the H.26x series and the MPEG series, follow the compression steps described above, with minor differences in details such as UV subsampling or entropy encoding.

2.1.2 H.264 Overview

H.264 is a joint MPEG and ITU-T video encoding standard [82]. It is also called MPEG-4 part 10 Advanced Video Coding (AVC). It is reported to have 50% bit-rate savings compared to H.263+ or MPEG-4 Advanced Simple Profile (ASP).

While the overall structure of an H.264 encoder is similar to that of other DCT-based algorithms, there are many improvements in the details. First, it allows finer-grained predictive coding. Motion estimation can be done on any 4×4, 8×8, 4×8, 8×4, 16×8, and 8×16 blocks in a macroblock. Second, it allows more than one reference frame, which includes B frames. Third, it adds intra spatial prediction in which a reference block is in the same frame as the block being predicted. Finally, the primary transform block size in H.264 is 4×4 instead of 8×8 as in most video coding standards. In summary, these changes decrease the unit size for prediction, extend the range of reference frames, and expand the modes for searching so the precision of prediction is improved. The major advantage is improved compression ratios. The disadvantages

are the extremely high computational cost for motion estimation and increased memory space for storing extra reference frames at the encoder and the decoder.

2.2 Video Adaptation

We will present the existing video adaptation technologies in two parts: the tailoring mechanism to provide a required video stream and the adaptation mechanism to determine what is required according to network conditions.

2.2.1 Tailoring Mechanisms

In this subsection, we first summarize tailoring mechanisms in general. We will then discuss tailoring mechanism for different quality dimensions.

2.2.1.1 A taxonomy of tailoring mechanisms

In general, the goal of tailoring mechanisms is to make a compressed video stream fit within the bit-rate allowed by the available network bandwidth and the receiver resource capability. The intuitive way to make a compressed stream fit a target bit-rate is to alter the video such as dropping every other frame and/or adjusting encoding parameters such as the quantization step.

Different types of video applications require different tailoring mechanisms. Tailoring mechanisms for live video applications are pretty straightforward because video is encoded at transmission time and the target bit-rate is known while the video is still uncompressed; the raw video can be altered and encoding parameters can be set accordingly. In contrast, tailoring mechanisms for stored video applications are more

Table 2-1 A taxonomy of tailoring mechanisms for stored video

Video representation	A non-scalable stream		Multiple non-scalable streams	A scalable stream
Mechanisms	Re-encoding	Transcoding	Multi-encoding	Scalable encoding
How to tailor	Change encoding parameters		Switch between streams	Add or drop layers
	Tailor raw video	Tailor video in the DCT domain		
Advantages	<ul style="list-style-type: none"> • Low storage cost • Low computational cost at encoding time 		<ul style="list-style-type: none"> • Low computational cost at transmission time 	
	<ul style="list-style-type: none"> • Fine-grained 	<ul style="list-style-type: none"> • May provide good trade-offs between computational cost and compression efficiency 	<ul style="list-style-type: none"> • Good compression efficiency 	<ul style="list-style-type: none"> • Good bandwidth efficiency for multi-casting
Disadvantages	<ul style="list-style-type: none"> • High computational cost at transmission time 	<ul style="list-style-type: none"> • Special algorithms required • Limited working range 	<ul style="list-style-type: none"> • Coarse-grained • High computational cost at encoding time 	
			<ul style="list-style-type: none"> • High storage cost 	<ul style="list-style-type: none"> • Special algorithms required • Compression efficiency overhead

complicated because stored video is usually already in a compressed format. We will focus primarily on tailoring already compressed video in this dissertation.

We divide tailoring mechanisms for stored video into several categories, as shown in Table 2-1, according to how the video is stored: (i) in a non-scalable stream, (ii) in multiple non-scalable streams, or (iii) in a scalable stream.

If the video is stored in one non-scalable stream, two types of tailoring mechanisms are available. They are *re-encoding* and *transcoding*. Re-encoding fully decodes the compressed video, alters the decompressed video in the pixel domain, and

re-encodes the altered video with appropriate encoding parameters. Re-encoding can tune the video bit rate to precisely match the network bandwidth; however, it requires a lot of computation at transmission time. Transcoding tries to reduce the computational cost of re-encoding by partially decoding a video stream, altering the video in the DCT domain, and partially re-encoding it. Altering the video in the DCT domain is not as straightforward as altering video in the pixel domain and requires specially designed algorithms. These algorithms usually lower the compression efficiency and have a very limited working range. A comprehensive survey of existing transcoding techniques is presented in [76].

If the video is stored in multiple non-scalable streams, tailoring is accomplished by switching between encoded streams. We refer to such an approach as *multi-encoding* in this dissertation. Multi-encoding spends a lot of time in encoding and a lot of space to store the compressed streams. Due to the limitation of encoding time and storage capacity, usually only a few such encodings are used at a given time. However, for each supported bit-rate, multi-encoding often has better compression efficiency than other mechanisms. Multi-encoding is currently being used by the IntelliStream system [3] from Windows Media and the SureStreams system [12] from Real Networks.

If the video is stored as a scalable stream, tailoring is accomplished through adding or dropping layers in the scalable stream. A scalable stream is generated by algorithms that structure a compressed stream into a base layer and several dependent

enhancement layers, which we refer to as *scalable encoding*. Scalable encoding usually has worse compression efficiency than non-scalable encoding; moreover, existing implementations support only two target rates. However, when more than one target bit-rate is required, scalable encoding allows the base layer to be shared by those targets so it can improve bandwidth efficiency if the underlying network provides group networking protocols such as multicast. Scalable encoding is included in many video compression standards such as MPEG-2.

2.2.1.2 Adaptation dimensions and tailoring mechanisms

In the previous subsection, we discussed ways in which a compressed video stream can be tailored to fit a target bit-rate. Changing the bit-rate of a video stream inevitably affects the video quality in one or more *dimensions*. The most common quality dimension is the actual visual quality of the individual video frames, which is commonly referred to as spatial fidelity and can be altered through changes in quantization. In addition, the frame rate, the spatial resolution, the cropped region, and the color fidelity can also be affected. For some applications, changing the bit-rate of a video stream can also be accomplished by selectively encoding part of the video because the quality or the utility of video depends on the content of the video. For example, for security surveillance applications, video that catches suspicious activities or subjects is useful; for habitat monitoring, video containing research subjects is useful. In summary, video can be tailored in a number. Below we briefly discuss techniques for tailoring video in different dimensions.

For re-encoding and multi-encoding, video is altered in the pixel domain. Altering video in the pixel domain is straightforward because there are no dependencies among frames and there are many algorithms available. For example, changing the frame rate requires only that frames be dropped before encoding; resolution scaling can be done through pixel sub-sampling, pixel interpolation, and filtering; algorithms for object identification and feature extraction can be used for content-based tailoring or filtering.

Transcoding requires altering video in the DCT domain. While operations in the DCT domain can approximate operations in the pixel domain, they introduce *drift errors* in predicative-encoded frames because the reference used during decoding may be altered and be different from the reference frame used during encoding. Algorithms for changing the spatial fidelity, the resolution, or the frame rate in the DCT domain have been proposed. Changing the spatial fidelity level in the DCT domain is relatively easy since it can be accomplished by changing the quantization parameters; however, to achieve good compression efficiency, it is necessary to recalculate the prediction errors in the DCT domain based on the altered reference to reduce the drift errors [76]. Changing the resolution is not that straightforward for block-based compression because the blocks are different at a new resolution, which means the old motion vectors and the old DCT coefficients are typically invalid. Algorithms for constructing new motion vectors and new DCT coefficients from the old ones in the compressed domain have been studied [2][46][47][88]. They are often designed to support downcaling to one lower resolution, which makes these algorithms unsuitable for supporting wide-range fine-grained multi-resolution video.

Changing the frame rate can be done by simply dropping frames along the dependency chain in that remaining frames are likely to be distributed unevenly along the time line. If a smooth frame rate is preferred, the dropped frames need to be evenly distributed, which breaks the old dependency and introduce drift errors. In this case, motion vectors may need to be re-estimated and prediction errors re-calculated in the DCT domain to reduce drift errors.

Scalable encoding algorithms in different dimensions have been studied. Algorithms for spatial fidelity scalability (usually called SNR scalability in research literature of multimedia) and for temporal scalability (supporting multiple frame rates) are mostly used and are included in video standards such as H.263, MPEG-2, and MPEG-4. H.263 and MPEG-2 also include algorithms for spatial scalability (supporting multiple resolutions). Dugad and Ahuja have proposed another spatial scalability scheme based on non-scalable encoders; this is referred as *Dugad's scheme* in this dissertation. Isolated regions in H.264 [81][82] and selective enhancement for MPEG-4 [69] can be used to encode ROIs; these schemes can be combined with multi-resolution video to better accommodate large variation in resolution. The scalability in a compressed stream is usually coarse, with one base layer and only one enhancement layer. One exception is the Fine Granularity Scalable (FGS) coding and Progressive Fine Granularity Scalable (PFGS) coding in MPEG-4[48][84], which provide fine-grained adaptation in the spatial fidelity dimension.

2.2.2 Adaptation Mechanisms

Adaptation mechanisms are coupled to the tailoring mechanisms being used. For re-encoding and transcoding, adaptation mechanisms decide when and how to change the video encoding parameters [42]. For multi-encoding, they decide when to switch and which stream to switch to [12][72][73]. For scalable encoding, they decide when to drop or add layers [17][54][64]. Because lower layers are always needed by higher layers, the decision of how many layers to send can be postponed until lower layers have been sent and a better estimation of network conditions becomes available. Still the decision needs to be made within a time window because the sending of higher layers should meet the latency requirement of an application for continuous playback. Feng [20], Kang[43], Krasic [44], and Miao [55] have proposed algorithms for window-based scheduling. The time window smoothes fluctuations of the video data rate and the network bandwidth at the cost of increased latency. For collection applications, the window can be very large because such applications typically do not have stringent latency requirements.

Despite their differences, most existing adaptation mechanisms target streaming applications with a maximum latency requirement, are based on IP-style networks with end-to-end connections, and are designed for streaming video from one source to one or more receivers.

2.3 Sensor Networks and Video Sensors

In this subsection, we describe the network conditions in a sensor network to help understand the challenges and our assumption for video collection. We then describe current video sensor platforms and video-based sensor applications.

2.3.1 Sensor Networks and Multi-hop routing

Sensor networks consist of smart sensors capable of sensing, computation, and communication [1][6][16][59]. They can be deployed in an ad hoc manner at places without networking infrastructure or power facilities. One challenge in building an operational sensor network is for sensors to self-organize to form multi-hop routes to store-and-forward data to a base station. Usually a sensor is not directly connected to the base station either because the distance between them is out of communication range or because the multiple short hops are more energy efficient than a long hop. Multi-hop routes are not always connected, either because of environmental factors or because of a TDMA MAC layer [87] used to save energy. There is no end-to-end connection in a sensor network and video adaptation in such a network has not been studied.

Many routing algorithms [7][28][41][83][85] have been proposed to set up multi-hop routes in sensor networks. In this dissertation, we assume that multi-hop routes have been setup and are relatively stable.

2.3.2 Video Sensors and Video-based Sensor Applications

As sensor hardware develops, a class of “large” sensor nodes that are capable of capturing and processing multimedia data such as audio and video have become available [31][32]. One representative example is the *Crossbow Stargate* device, which has a 400MHz Intel X-scale processor, up to 64 megabytes memory and one gigabyte flash memory, and an 802.11 wireless interface that can provide wireless bandwidth from 500Kbps to 10Mbps. These sensor nodes are much more powerful than typical “small” sensor nodes such as Berkeley motes [33] and make it possible to capture multimedia information in addition to scalar data. The Panoptes video sensor [19] is built on this platform and is used for our experiments. The research effort towards building applications upon video sensor networks has just started [21]. Examples include Panoptes [19], SensEye [45], the CVSBN project [27], and distributed attention [10].

CHAPTER 3

***BONNEVILLE*: SUPPORTING WIDE-RANGE FINE-GRAINED MULTI-RESOLUTION VIDEO**

As the diversity of video devices increases, video adaptation systems will need to support adaptation over an extremely large range of display requirements (e.g. 90×60 to 1920×1080.) In this chapter, we examine tailoring techniques for resolution scaling to support the adaptation. We believe that *Bonneville*, a combination of multi-encoding and scalable encoding/transcoding, is necessary to accommodate large variation in resolution.

3.1 Introduction

In chapter 1 we presented an application scenario in which high-resolution video is streamed to display devices of different sizes, and resolution adaptation is needed to adapt video over a wide-range of resolutions. In this section, we present more application examples that require wide-range fine-grained multi-resolution video. We then propose *Bonneville*, a framework to structure stored video to support many resolutions.

3.1.1 More Motivating Examples

In addition to adapting to display requirements, resolution adaptation also allows the video stream to be adapted to underlying networking constraints. Currently,

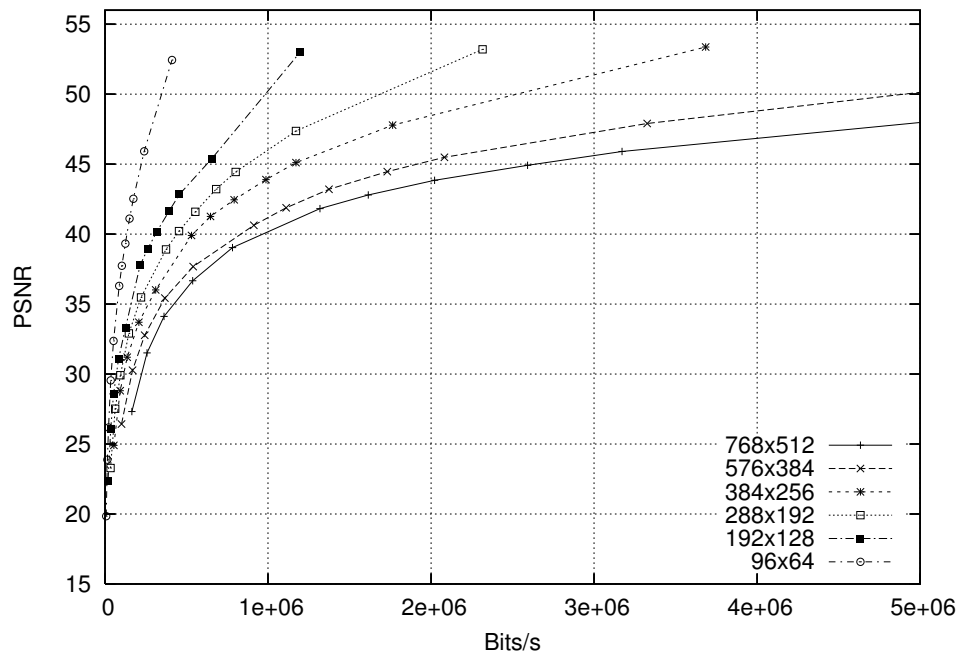


Figure 3-1 Encoding *The Italian Job* at different resolutions. This figure shows average bit-rates and average PSNRs of video streams encoded at different resolutions with different quantization scales. Each line represents a resolution and dots on that line represent different quantization scales. Each dot in this figure represents a feasible bit-rate.

spatial fidelity adaptation (also called SNR adaptation in research literature of multimedia) is often used to deal with insufficient bandwidth. When the bandwidth is extremely low, spatial fidelity adaptation cannot reduce the stream size enough and resolution adaptation has to be used to reduce the number of pixels to be encoded. To demonstrate this, we have encoded 300 frames from the movie *The Italian Job* using the reference codec of H.264 [29] at different resolutions with different quantization scales as shown in Figure 3-1. When we encode at resolution 768x512, the smallest stream size we encode by adjusting the spatial fidelity (using the largest quantization scale 51) is 164Kbps; by reducing the resolution to 384x256, the stream size can be further reduced to 54Kbps. Even if spatial fidelity adaptation can make the target

bandwidth, downscaling the resolution can result in better perceptual quality than lowering the spatial fidelity in some cases. Figure 3-2(a) and Figure 3-2(b) show a decoded frame from two streams encoded at 768×612 and 384×256 respectively, both having a bit-rate of 164Kbps. The decoded picture from the 384×256 stream shown in Figure 3-2(b) obviously has better picture fidelity. For example, we can see the back of a truck on the left side of the picture in Figure 3-2(b) while in Figure 3-2(a) it is so blurred that it is hardly distinguished from the buildings in front of it. In summary, adjusting the resolution extends the range and dimensions for bandwidth adaptation; adjustment over a wide range is needed to fit possible bandwidth limits and the user's needs.

Resolution adaptation, combined with ROI adaptation, is also useful to support Pan-Tilt-Zoom-like operations for users to navigate through high-resolution video over best-effort networks. Transmission bandwidth and remote computation can be saved by sending the video at the viewed resolution instead of the high-resolution video; and potentially the video may be viewed at many resolutions that require fine-grained resolution adaptation.

3.1.2 Proposed Approach

In order to support resolution adaptation, re-encoding, transcoding, multi-encoding, or scalable encoding can be used. Each type of mechanisms introduces some overhead when supporting multiple resolutions. Re-encoding has high computational overhead; transcoding and scalable encoding have compression efficiency overhead; and multi-encoding has storage overhead. The problem of



(a) transmitted at 768×512



(b) transmitted at 384×256

Figure 3-2 The difference between lowering the spatial fidelity and downscaling the resolution. These two pictures are decoded from two H.264 streams. Both streams contain the same 300 frames from the Italian Job encoded by the H.264 reference codec. Stream (a) is encoded at the resolution 768×512 with a very low spatial fidelity; stream (b) is encoded at 384×256 but the spatial fidelity is higher. Both streams are about 164Kbps. Picture (b) is much sharper than picture (a).

extending one mechanism to support resolution adaptation over a wide range, if even possible, is that the accumulated overhead may become too large and make the mechanism unsuitable for practical use.

We propose to combine existing tailoring technologies; we believe that the combination of mechanisms can provide the best-balanced performance. In particular, we propose to combine multi-encoding with scalable encoding and transcoding. We call this hybrid framework *Bonneville*.

We will compare tailoring mechanisms in the *Bonneville* framework with other tailoring mechanisms. We group tailoring mechanisms into three architectures based on the number of “full” encodings that are used to represent the video on the server: the one-encoding-for-all-resolutions architecture, the one-encoding-per-resolution architecture, and the hybrid architecture. *Bonneville* is a hybrid architecture. We describe the three architectures in Section 3.2. We study the bandwidth efficiency, computational cost, and storage cost for these architectures in Section 3.3 and present an in-depth analysis of how *Bonneville* provides good architectural trade-offs in providing fine-grained wide-range multi-resolution video. Our work also provides guidelines to structure the multiple encodings within the *Bonneville* framework such as the number of encodings.

3.2 Tailoring Mechanisms for Resolution Adaptation

Despite rapid progress in storage capacity and transmission bandwidth, video compression is still a key technology for video applications because of large

compression ratios that are achievable. Currently, most compression techniques are either DCT-based or wavelet-based. Wavelet-based algorithms [53][89] perform a wavelet transform on the entire image, which results in a hierarchical representation of an image. In the hierarchy, each layer represents a frequency band, which corresponds to a resolution. Thus, wavelet-based compression supports multi-resolution video inherently. However, the ability to perform ROI adaptation is limited due to the wavelet transform. We believe that both ROI adaptation and resolution adaptation are needed to accommodate large variation in resolution. In DCT-based compression, ROI cropping can be supported by flexible macroblock ordering [81][82] or bit-plane shifting[69]. While somewhat challenging to support multi-resolution video for DCT-based compression, its adoption into standards such as the MPEG series and the H.26x series coupled with the ability to support ROI adaptation make it an interesting technique to use.

In the remainder of this section, we describe DCT-based tailoring mechanisms for resolution adaptation as well as their advantages and disadvantages for supporting fine-grained wide-range adaptation.

3.2.1 A Single Encoding for All Resolutions

A very simple architecture to provide wide-range fine-grained video resolution adaptation is to *encode the highest resolution once* and generate all other resolutions from the one encoding. Under this architecture, resolution adaptation is accomplished using one of the following two approaches.

The first approach is to spend more time encoding the stream so it is more amenable to resolution downscaling; for example, scalably encoding the stream with a base layer and multiple enhancement layers. We note here that the use of the term *enhancement layer* usually means higher spatial fidelity. For our purposes, we use enhancement layer to refer to layers that provide higher resolutions. Examples of scalable encoding include the MPEG-2 spatial scalability scheme [30], where an upscaled base layer provides extra references for motion estimation, and Dugad's spatially scalable encoder [15] constructed from non-scalable encoders, where an enhancement layer encodes differential signals between the high resolution images and the upscaled base layer. In addition to ease of scaling, it can save bandwidth in multicast scenarios because low resolutions are included in high-resolution streams. However, scalability is not free because the compression efficiency of scalable encoding is lower than that of non-scalable encoding. If scalable encoding is extended to support many resolutions, the overhead accumulates and can become significant. We will evaluate the compression efficiency of scalable encoding for wide-range fine-grained multi-resolution video.

The second approach to provide resolution adaptation is to compress the stream with minimal extra information and spend more time scaling the video stream to a different resolution as needed. Scaling can involve (i) a full re-encoding where the stream is decompressed and recompressed, or (ii) a partial re-encoding (transcoding) where the stream is altered in the compressed domain [14][37][75]. Different

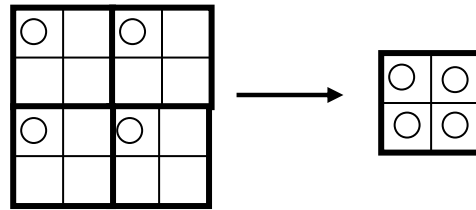
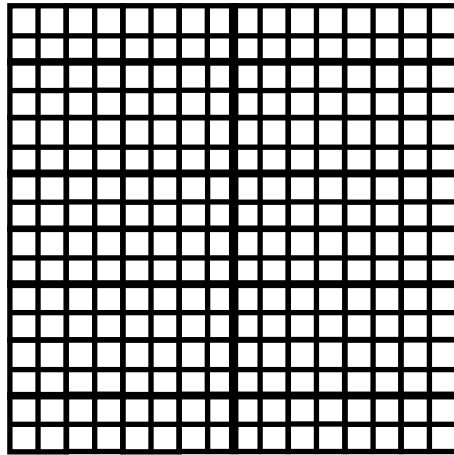


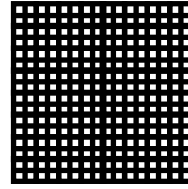
Figure 3-3 Generate DCT coefficients for low-resolution image in the DCT domain.
 The intuition behind many algorithms is to extract 4x4 low-frequency coefficients of the MxN video from four adjacent blocks to form the 8x8 coefficient-matrix of the M/2 x N/2 video.

transcoding algorithms need different amounts of work when a client requires a smaller resolution; but in general, they need less work than re-encoding.

There are two groups of transcoding algorithms. The first group of algorithms tries to derive new DCT coefficient matrices for low-resolution images. For example, many algorithms extract 4x4 low-frequency coefficient from four adjacent blocks in the high-resolution video to form one 8x8 coefficient matrix for one block in the low-resolution video as shown in Figure 3-3. Thus, these algorithms can derive only one low resolution that has to be one-fourth of the high resolution (half in each dimension). The second group of algorithms simply drops DCT AC coefficients at the server side and the downscaling is done at the client side. The working range of this class is also very limited because the compression efficiency for the low-resolution video degrades rapidly when the number of pixels in the low-resolution video approaches or becomes less than the number of blocks in the high-resolution video. Figure 3-4 provides an example of the degradation of compression efficiency. It shows the Y component of a 128×128 -pixel image, which is divided into 256 8×8 -blocks. After the DCT



The original image: 128x128 pixels,
16x16 8x8-blocks



The downscaled image through transcoding:
16x16 pixels, 16x16 1x1 blocks



The Downscaled image through re-
encoding: 16x16 pixels, 2x2 8x8-blocks

Figure 3-4 Comparison of compression efficiency between re-encoding and transcoding by dropping AC coefficients. In each 8x8 block, about 70% coefficients are zeros after DCT transform and quantization. A compressed 8x8 block is much smaller than 64 1x1 block.

transform and quantization, 30% of the coefficients are typically non-zero; therefore, each pixel is represented by 0.3 compressed coefficients. Suppose we are to downscale the image to 16x16 pixels. If the image is re-encoded, the 16x16 pixels are divided into four 8x8-blocks and a similar compression ratio can be retained. If we simply drop AC coefficients, the 256 blocks remain with one DC coefficient in each block. Thus, one pixel is represented by one compressed coefficient and the compression efficiency is about 3.3 times worse than re-encoding. More importantly, this is the lower bound of the number of blocks. If a smaller resolution is required, the 256 blocks still need to be encoded. In summary, transcoding alone cannot support many resolutions efficiently.

In addition to the two traditional approaches, we also propose to mix these two approaches by scalably encoding the video once and then transcoding enhancement layers to generate resolutions between layers. This mixed approach effectively takes the advantage of the positive aspects of each while trying to avoid the disadvantages. The performance of this mixture is unknown and we will figure out through experiments whether the mixture can provide good trade-offs between coding efficiency, computational cost, and storage cost.

3.2.2 One Encoding Per Resolution

Another simple architecture to provide multi-resolution video is to encode as many streams as resolutions that are required. In this way, each encoding has been optimized for a particular display (or at least one that is of similar resolution). However, the optimal per-resolution efficiency does not lead to optimal overall bandwidth efficiency because different resolutions do not share data. Another drawback is the computational overhead involved in creating and managing potentially many streams since there are a large number of different display sizes as shown in Table 3-1. For stored systems, this is further complicated by the fact that the resolutions required may not be known a priori. Thus, some form of adaptation may always be necessary.

For this dissertation, we will use the one-encoding-per-resolution architecture in the experiments to provide a baseline for how well one could have done for a particular resolution (quality as well as bandwidth requirements).

Table 3-1 Available display sizes

Cell Phones	PDAs	Laptops	Top-of-the-line Monitors
96x36	160x160	640x480	1680x1050
96x65	160x240	800x600	1920x1200
101x80	240x100	1024x480	2048x768
128x128	240x200	1024x768	2048x1536
160x128	320x240	1280x800	2560x1600
208x176	320x320	1280x1024	
240x160	480x160	1400x1050	
320x208	480x320	1440x900	
320x240	640x240	1600x1200	
640x200	800x480		
640x320	800x600		

3.2.3 Hybrid Architectures and *Bonneville*

In addition to the architectures described in the previous two subsections, one can encode several candidate resolutions that cover a class of displays and then create all other resolution streams from the encoded streams. In effect, this hybrid architecture combines the above two architectures. The goal is to extend the working range of a single architecture and to provide better trade-offs in bandwidth efficiency, computational cost, and storage cost.

In the hybrid architecture, resolutions are divided into groups and each group has one full encoding. In this dissertation, we assume that all groups use the same tailoring mechanism to support resolutions in that group. For example, all groups might be based on scalable encoding and use the same scalability scheme; or all groups might be based on non-scalable encoding and use the same transcoding algorithm.

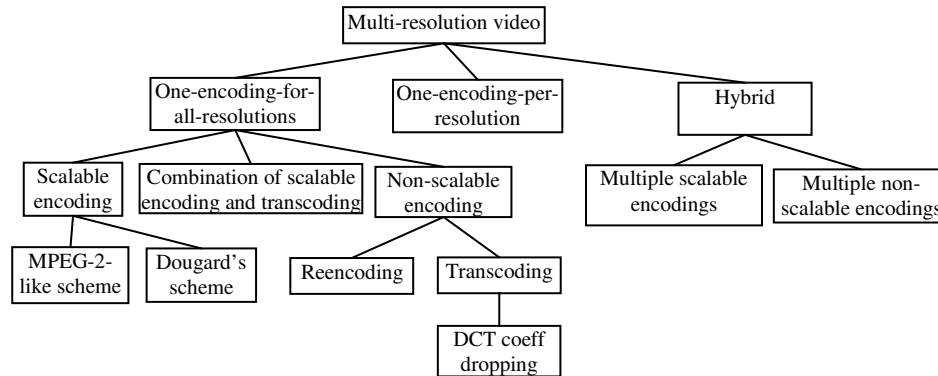


Figure 3-5 Multi-resolution mechanism summary. There are three possible architectures: one-encoding-for-all-resolutions, one-encoding-per-resolution, and the hybrid architectures. To generate multiple resolutions from one encoding, there are four possible mechanisms: scalable encoding, re-encoding, transcoding, and the combination of scalable encoding and transcoding. Examples of scalable encoding schemes and transcoding techniques are shown.

Bonneville is a hybrid architecture with multiple scalable encodings. Each group contains a scalable encoding; some resolutions in the group may not be included in that encoding and these resolutions between layers are generated by transcoding.

3.2.4 Mechanism Summary

The mechanisms that can be employed to support multi-resolution video are summarized in Figure 3-5. The question is how should one structure the video to support such adaptation to a large number of display sizes. Intuitively, we believe that supporting such video will fall into the hybrid architecture category because it allows the efficient trade-off between computation for encoding and computation for display-dependent streaming.

3.3 Experiments and Analysis

In this section, we will present a number of experiments to highlight the various trade-offs one can make in supporting multi-resolution video. We first present our

experimental setup and the metrics we use to compare mechanisms under different architectures. We then present the results for bandwidth efficiency, computational cost, and storage cost in the following three subsections. Finally, we discuss guidelines for designing a system for multi-resolution video in the framework of *Bonneville*.

3.3.1 Experimental Setup



In this subsection, we describe how we set up the test sequences, the testing resolutions, the codec, the encoding parameters of the codec, the transcoding and scalable encoding algorithms, and the algorithms for spatial scaling in the following subsections.

3.3.1.1 Test sequences and testing resolutions

We use the test sequence *Bus*, which is a standard test sequence in the research community, to test scalable encoding parameters and spatial scaling algorithms. However, *Bus* is at CIF resolution (352×288) and the resolution is not high enough to test resolution adaptation over a wide range as we need in this dissertation.

We choose two video clips with higher resolutions as test sequences for resolution adaptation, one motion-intensive and the other non-motion-intensive so our results are not biased towards either. The motion-intensive clip consists of the 300 frames from *The Italian Job*. The non-motion-intensive clip is a 100-frame 3008×2000 video sequence we took at a *Street Corner*. As we will discuss in the following section, the codec we choose is the H.264 reference codec, which accepts only those resolutions

Table 3-2 Testing sequences and resolutions

Name of video sequences	A sample picture	The original resolution	Testing resolutions
<i>The Italian Job</i>		720×480	768×512, 576×384, 384×256, 288×192, 192×128, 96×64
<i>Street Corner</i>		3008×2000	2880×1920, 2160×1440, 1920×1280 1440×960, 960×640, 720×480, 480×320, 384×256, 288×192, 192×128, 96×64

that are multiples of 16 in each dimension. Therefore, we padded *The Italian Job* images to 768×512 and cropped the *Street Corner* images to 2880×1920. We took the raw video and converted it into YUV420 and then downsized the raw images to small resolutions. The downsized image sequences are used as the reference for that resolution when calculating PSNRs (Peak Signal Noise Ratio). PSNR is the metric we use to measure video quality as we will discuss later in Section 3.3.2.

All the testing resolutions we chose, along with a sample frame from each sequence, are listed in Table 3-2. Some resolutions chosen are non-standard because of the multiple-of-16 restriction, yet they are close to sizes of all kinds of display

devices. For example, 96×64 is close to the size of many cell phones; 480×320 is a typical size for PDA screens; 960×640 and 1440×960 could be the resolutions for laptops or desktops monitors; 1920×1280 is almost the resolution for HDTV; 2160×1440 and 2880×1920 can be used for top-of-the-line monitors. All the testing resolutions together also provide a wide range of bit-rates that could fit various network conditions.

3.3.1.2 The base codec and encoding parameters

We constructed our experiments mainly based on the H.264 reference software [29], which focuses on compression efficiency. The transcoding algorithms and scalable encoding algorithms used in the experiments are implemented based on this codec and will be described in the following sections.

Figure 3-6 shows part of the encoding parameters we use for the H.264 codec. *IntraPeriod* 4 means there are three P frames after each I frame; *FrameSkip* 2 means there are two B frames between any I or P frames. Thus, the GOP structure is IBBPBBPBBPBB. We allow three reference frames for motion estimation and allow all inter and intra prediction modes. One important parameter not shown is the quantization scale. We use the same quantization scale for all frames (the H.264 codec allows different quantization scales for I, P, and B frames) and the same quantization scale for the Y component and the UV components. The range of legal quantization scales is 1 to 51. The quantization scales we chose are 8, 16, 20, 22, 24,

```

# Files
InputHeaderLength    = 0      # If the inputfile has a header
StartFrame           = 0      # Start frame for encoding. (0-N)
FramesToBeEncoded    = 300    # Number of frames to be coded
FrameRate            = 25     # Frame Rate per second (0.1-100.0)
TraceFile            = "trace_enc.txt"
ReconFile            = "test_rec.yuv"
OutputFile           = "ij.h264"

# Encoder Control
ProfileIDC           = 77    # Profile IDC (66=baseline, 77=main,
                             # 88=extended; FREXT Profiles: 100=High,
                             # 110=High 10, 122=High 4:2:2, 144=High 4:4:4)
LevelIDC             = 50    # Level IDC (e.g. 20 = level 2.0)

IntraPeriod          = 4     # Period of I-Frames (0=only first)
IDRIntraEnable       = 0     # Force IDR Intra (0=disable 1=enable)
FrameSkip            = 2     # Number of frames to be skipped in input
                             # (e.g 2 will code every third frame)

ChromaQPOffset       = 0     # Chroma QP offset (-51..51)
UseHadamard          = 1     # Hadamard transform (0=not used, 1=used)
SearchRange          = 16    # Max search range
NumberReferenceFrames = 3     # Number of previous frames used for inter
motion search (1-5)
PList0References     = 0     # P slice List 0 reference override (0
                             # disable, N <= NumberReferenceFrames)
Log2MaxFrameNum      = 0     # Sets log2_max_frame_num_minus4 (0-3:based on
                             # FramesToBeEncoded, >3:Log2MaxFrameNum - 4)
MbLineIntraUpdate    = 0     # Error robustness(extra intra macro block
                             # updates)(0=off, N: One GOB every N frames
                             # are intra coded)
RandomIntraMBRefresh = 0     # Forced intra MBs per picture
InterSearch16x16     = 1     # Inter block search 16x16 (0=disable, 1=enable)
InterSearch16x8      = 1     # Inter block search 16x8 (0=disable, 1=enable)
InterSearch8x16      = 1     # Inter block search 8x16 (0=disable, 1=enable)
InterSearch8x8       = 1     # Inter block search 8x8 (0=disable, 1=enable)
InterSearch8x4       = 1     # Inter block search 8x4 (0=disable, 1=enable)
InterSearch4x8       = 1     # Inter block search 4x8 (0=disable, 1=enable)
InterSearch4x4       = 1     # Inter block search 4x4 (0=disable, 1=enable)

IntraDisableInterOnly = 0    # Intra modes for Non I-Slices
Intra4x4ParDisable   = 0     # Vertical & Horizontal 4x4
Intra4x4DiagDisable  = 0     # Diagonal 45degree 4x4
Intra4x4DirDisable   = 0     # Other Diagonal 4x4
Intra16x16ParDisable = 0     # Vertical & Horizontal 16x16
Intra16x16PlaneDisable = 0   # Planar 16x16
ChromaIntraDisable   = 0     # Intra Chroma modes other than DC

UseFME               = 0     # Use fast motion estimation (0=disable, 1=enable)

```

Figure 3-6 The H.264 encoder configuration file.

26, 28, 33, 37, 41, 45, and 51. Usually quantization scales in the twenties are practical and the quantization scale 24 is used in most of our experiments unless otherwise specified.

3.3.1.3 Transcoding

We have modified the H.264 codec to drop AC coefficients as the transcoding scheme. Since the H.264 codec uses 4×4 DCT, the working range of this scheme is very limited. Suppose the high resolution is $M \times N$, resolutions $3M/4 \times 3N/4$, $M/2 \times N/2$, and $M/4 \times N/4$ can be derived by keeping the 3×3 low-frequency coefficients, the 2×2 low-frequency coefficients, and the DC coefficients, respectively. The downscaling is done at the client side after decoding using the *sinc* filter from *ImageMagic* [39].

3.3.1.4 Scalable encoding

For our experiments, we will test two spatially scalable encoding schemes. One is the scheme used in MPEG-2, which is referred to as the *MPEG-2-like* scheme in this chapter; the other is Dugad's scheme that is based on a non-scalable codec. We first describe how these schemes are implemented based on the H.264 codec and are extended to support more than one resolution. Then we discuss the representation of differential signals in Dugad's scheme and the choice of scaling algorithms.

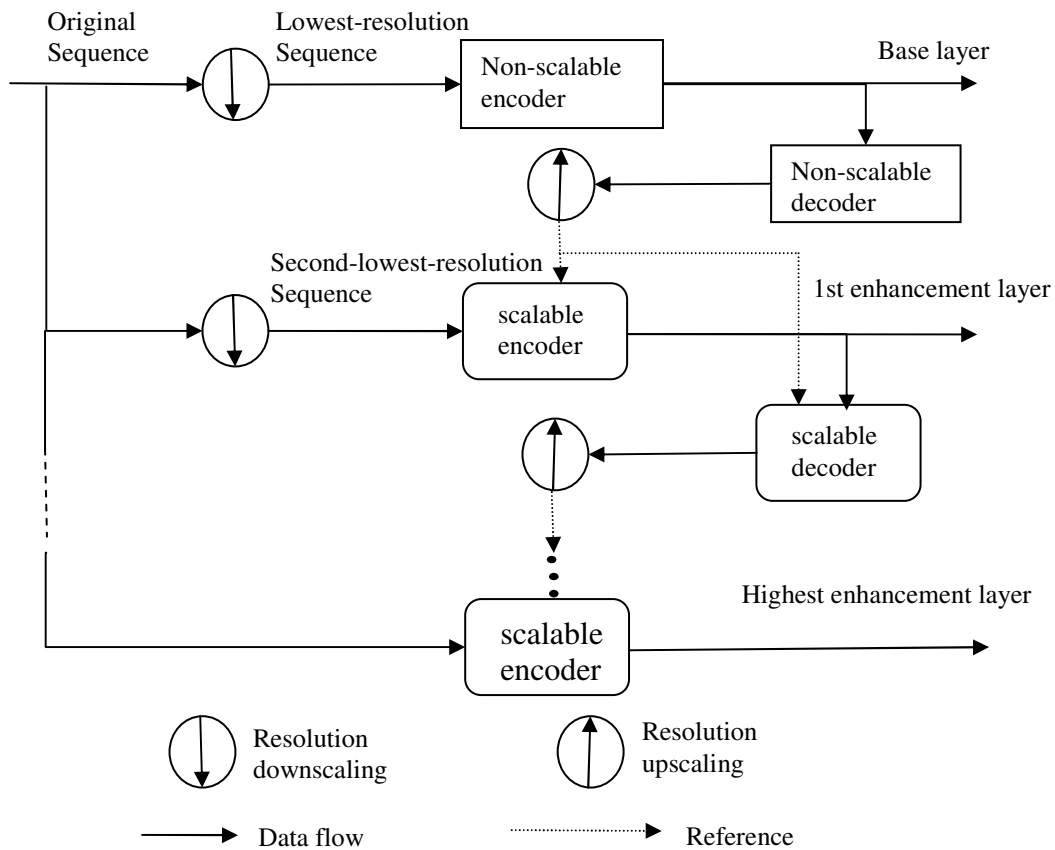


Figure 3-7 The MPEG-2-like spatially scalable encoder.

3.3.1.4.1 MPEG-2-like scheme

We have added MPEG-2-like spatial scalability into the H.264 codec and implemented an H.264 scalable encoder, as shown in [13]. The scalable encoder has four more reference frames for motion estimation in addition to the regular three temporal reference frames. One is the upscaled decoded low-resolution frame, which is called the spatial reference frame; the other three are the averages of the spatial reference frame and the three temporal reference frames. The scalability scheme is extended to generate more than two layers, as shown in Figure 3-7.

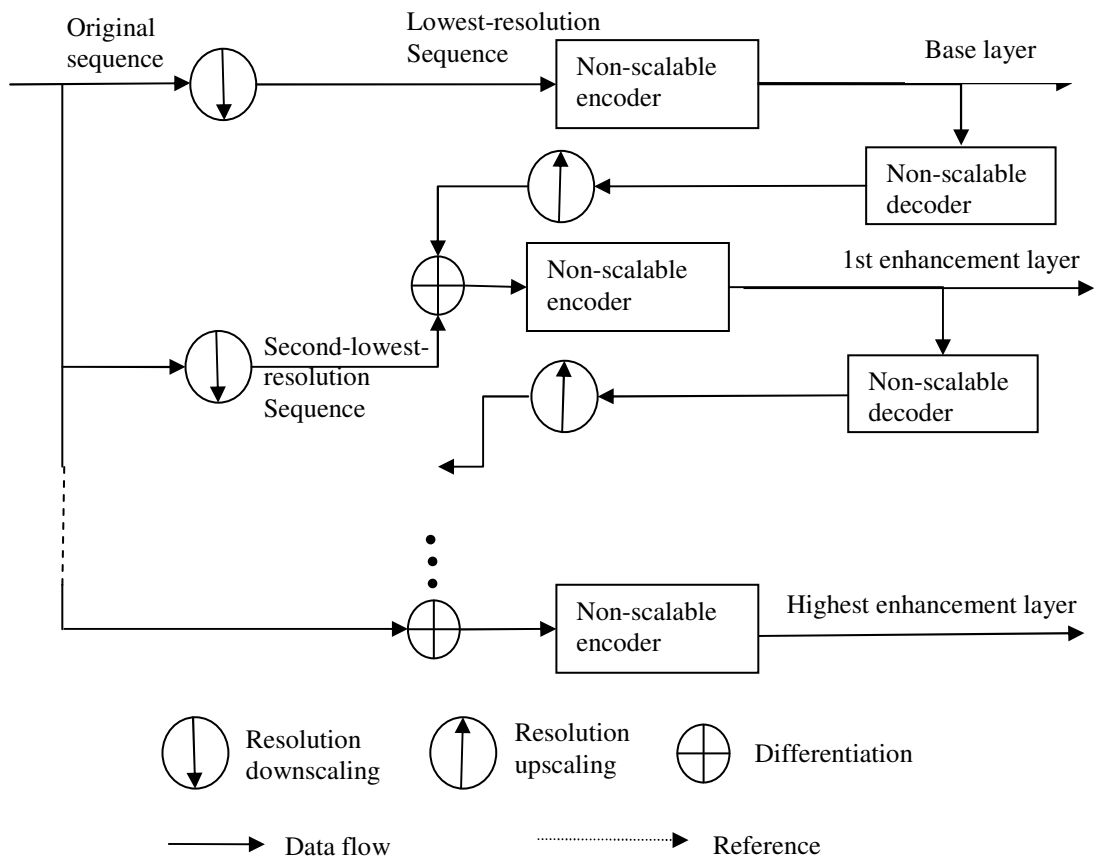


Figure 3-8 Dugad's spatially scalable encoder.

3.3.1.4.2 Dugad's scheme

We have constructed a spatially scalable encoder according to Dugad's scheme [15], in which differential signals between the high-resolution video and the upsampled decoded lower resolution video is encoded for enhancement layers. We have extended the scheme to more than two layers as shown in Figure 3-8. The construction of such an encoder is simple because it is based on non-scalable encoders.

3.3.1.4.3 Representation of differential signals in Dugad's scheme

In Dugad's scheme, enhancement layers encode differential signals, which are supposed to be smaller than encoding the high-resolution video directly. The differential signals are represented in the same YUV420 format, in which each value is represented by eight bits ranging from 0 to 255, as in the original video. However, differential values range from -255 to 255 instead of 0 to 255, thus doubling the range of values that we need to represent in eight bits. We need to squeeze the range to [-127, 128] and we consider two ways. One way is to divide the differential signals by 2; the other way is to truncate the range by making values less than -127, -127 and those larger than 128, 128. The first way introduces rounding errors in about half of the pixels. The second way introduces overflow errors which are the difference between -127 and a differential value less than -127 or the difference between 128 and a differential signal larger than 128. An overflow can be as large as 128 and its contribution to the Mean Square Error and the PSNR could be equivalent to rounding errors at 16,384 pixels. Since the upscaled decoded lower resolution video should be close to the high-resolution video, we believe that the chance for overflow is small and the "truncate" representation could achieve better video quality for enhancement layers.

To confirm that the "truncated" representation can achieve better PSNRs than the "divided by 2" representation, we ran the encoder on the *Bus* test sequence using the two representations. The base layer is at QCIF (176×144) and the enhancement layer is at CIF (352×288). Figure 3-9 shows the average PSNRs of the reconstructed CIF

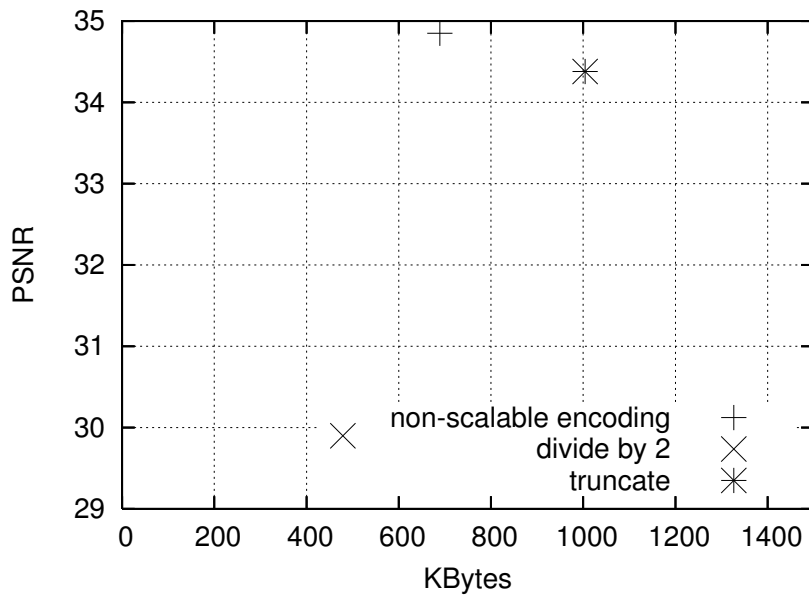


Figure 3-9 Comparison of different representations for differential signals. The PSNRs are for the *Bus* test sequence at CIF resolution. The base layer is at QCIF.

images and the sizes of the compressed streams when they are scalably encoded and when they non-scalably encoded. When the differential signals are in the “truncated” representation, the reconstructed images from the scalable stream have PSNRs close to the decoded images from the non-scalable stream and the size of the scalable stream is larger than the non-scalable stream. When the differential signals are in the “divided by 2” representation, the scalable stream is smaller than the non-scalable stream but the PSNRs of reconstructed images are about 5dB lower than those from the non-scalable stream. For this dissertation, we will use the “truncated” representation because when we compare the two schemes we expect them to have the same or similar PSNRs, especially given the fact that they represent the same image.

3.3.1.4.4 Choice of scaling algorithms

The resolution-scaling algorithm used in a scalable encoding scheme can greatly change its compression efficiency because decoded frames at a low-resolution need to be upscaled to a higher resolution to help reduce the bit-rate when encoding the higher resolution. For the MPEG-2-like scheme, an upscaled frame is a reference frame for motion estimation and a reference frame with good quality can potentially provide good matches to the frame being encoded thus reduce the size of the compressed frame. For Dugad's scheme, an upscaled frame with good quality can reduce the size of the differential signals thus reduce information to be encoded. Therefore, we need to choose an appropriate scaling algorithm so that the compression efficiency of the scalable encoding scheme is not limited by it.

There are three categories of scaling algorithms: pixel re-sampling, pixel interpolation, and transform-based scaling. We have tested all three categories, using the scaling functions in *ImageMagick* [39], an open-source image manipulation tool. The three categories correspond to the “-resample”, “-scale”, and “-resize” options in *ImageMagick*, respectively. For transform-based scaling, we have tested the *sinc* filter and the *Lanczos* filter.

Our experiments are mainly based on Dugad's scheme. The test sequence is the *Bus* sequence; the base layer is at QCIF and the enhancement layer is at CIF. The PSNR and stream size for the CIF resolution are shown in Figure 3-10. As shown the transform-based algorithms outperform other algorithms. The *Lanczos* filter and the *sinc* filter have very similar results, which are about 0.25 dB better than pixel

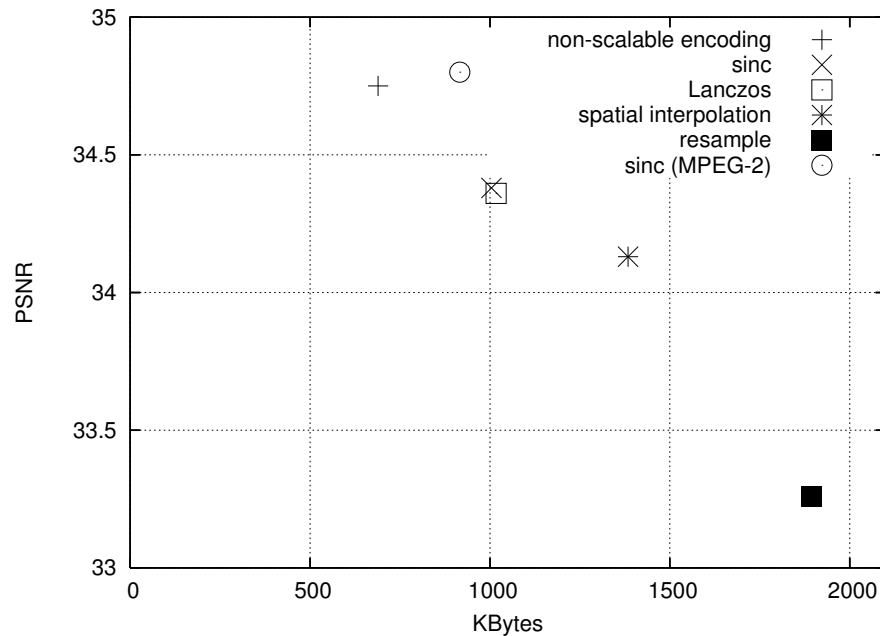


Figure 3-10 Comparison of scaling algorithms. The PSNRs are for the *Bus* test sequence at CIF resolution. The base layer is at QCIF.

interpolation and 1.12dB better than pixel resampling, and the stream size is 27.40% and 46.96% smaller. We also ran the MPEG-2-like algorithm using the *Lanczos* filter and the *sinc* filter and the results are satisfying, only 10% larger stream size and 0.1dB better quality compared to non-scalable encoding. Therefore, we use the *sinc* filter (because it is more common than the *Lanczos* filter) throughout the experiments.

3.3.1.5 Summary of experimental setup

For our experiments on multi-resolution video, we will use two test sequences: *The Italian Job* and *Street Corner*. The former is motion-intensive and the resolutions range from 96×64 to 768×512. The latter is non-motion-intensive but rich in details and resolutions range from 96×64 to 2880×1920.

The H.264 reference codec is our non-scalable encoder. It is also the base for implementing our transcoding and scalable encoding algorithms. For transcoding, we implement DCT-coefficient dropping; for scalable encoding, we implement the spatial scalability scheme in MPEG-2 and Dugad's scheme. We choose the *sinc* filter as the scaling algorithm for the two scalable encoding schemes because the compression efficiency is good when the *sinc* filter is used.

3.3.2 Metrics

For this dissertation, bandwidth efficiency, computational cost, and storage cost are the three metrics we use to compare approaches.

Bandwidth efficiency is determined by two factors: the bit-rate of a video stream and the video quality it presents. We have to compare both the bit-rate and the video quality when compare bandwidth efficiency. A video stream has better bandwidth efficiency than another stream in one of the three cases: (i) it has better video quality and lower bit-rate, (ii) it has the same quality as the other stream but has lower bit-rate, or (iii) it has the same bit-rate as the other stream but has higher video quality. If the stream has better (worse) video quality and higher (lower) bit-rate than the other stream, their bandwidth efficiency is not really comparable.

In our experiments, video quality is measured by the average PSNR (Peak Signal Noise Ratio) of the Y component of all frames. PSNR is the log of the ratio of the square of the peak signal (255 in an 8-bit system) to the MSE (Mean Square Error). Although PSNR is often criticized for having poor correlation with the human vision

system, it is the most widely used objective metric for video quality because it is simple and performs statistically equivalent to some more complicated schemes [65] such as the Just Noticeable Difference model from the Sarnoff Labs [50]. Video quality can be more accurately measured by subjective testing; however, it is too costly and time consuming and not adopted in our experiments. After all, neither objective nor subjective methods can measure video quality directly but provide an indication of how a degraded picture compares with a reference picture.

There are two ways to measure bandwidth efficiency, the per-resolution bandwidth efficiency and the overall bandwidth efficiency for all resolutions. The per-resolution bandwidth efficiency is very important for receivers to use their bandwidth efficiently. Overall bandwidth efficiency is very important for the server to improve its performance because it often needs to stream multiple resolutions at the same time. Per-resolution bandwidth efficiency is determined by its compression efficiency thus we use the two terms interchangeably. The overall bandwidth efficiency may not be the sum of the bandwidth requirement for each of the individual resolutions. Scalable encoding algorithms usually have lower per-resolution compression efficiency but have the potential for higher overall compression efficiency because different resolutions can share data (suppose that the underlying networking protocols can support data sharing.)

For computational cost, we consider the server side cost as the primary metric. Computational cost at encoding time and streaming time are both considered.

Computational cost at encoding time, however, usually has a looser time constraint thus is not as important as streaming time cost. It is difficult to measure computational cost, which depends on factors such as algorithms, compilers, CPUs, caching, and operating systems. Execution time is not a good measurement in our case since the H.264 reference codec we used takes about one day to encode one hundred 2880×1920 frames. This codec is not intended for any real applications. Therefore, the execution time of this codec does not have any practical meaning. Fortunately, our goal is to compare runtime computational cost among different approaches. For our purposes, we will use the number of DCTs, the number of IDCTs, and the number of motion estimations as an indication of the amount of work each algorithm needs to perform because they represent the most expensive computations for DCT-based video compression. For example, a H.261 encoder is reported to spend about 60% of computation in motion estimation and about 25% in DCT/IDCT[25]. The H.264 codec we use is likely to spend more time in motion estimation than other compression standards because for each macroblock, motion estimation can be done for different blocks within the macroblock over multiple reference frames as described in Section 2.1.2. In this chapter, we count the match-searching for one block over one reference frame as one motion estimation.

3.3.3 Experimental Results and Analysis

We will use the results from the one-encoding-per-resolution architecture as a reference in comparison. The one-encoding-per-resolution architecture is supposed to have (i) the best per-resolution bandwidth efficiency because each resolution is non-

scalably encoded and has optimized compression efficiency, (ii) the best streaming time computational cost because no computation is needed, (iii) the worst overall bandwidth efficiency because there is no data sharing among different resolutions, (iv) the worst encoding time computational cost because it has to encode every resolution, and (v) the worst storage cost because it stores a stream for every resolution.

Our experiments will show that *Bonneville*, multiple scalable encodings combined with transcoding, can efficiently balance many of the performance metrics. In particular, we believe that multiple scalable encodings with less than five layers in each encoding are a good start point and DCT-coefficient dropping can be used to generate one resolution between layers.

3.3.3.1 Bandwidth efficiency

We first show the limitations of one-encoding-for-all-resolutions (one-encoding for short) on per-resolution bandwidth efficiency and the improvement in the overall bandwidth efficiency compared to the one-encoding-resolution architecture. We then show how a hybrid architecture can improve the per-resolution bandwidth efficiency of the one-encoding architecture and retain its good overall bandwidth efficiency.

3.3.3.1.1 The one-encoding-for-all-resolutions architecture

The purposes of this subsection are (i) to find out the limitations when scalable encoding and re-encoding are pushed to support many resolutions and (ii) to find out whether the one-encoding architecture can improve the overall bandwidth efficiency compared to the one-encoding-per-resolution architecture. The algorithms being

compared are re-encoding, MPEG-2-like spatially scalable encoding, and Dugad's spatially scalable encoding. Since the H.264 codec uses a 4×4 DCT transform, DCT-coefficient dropping can only generate three lower resolutions from one full encoding so DCT-coefficient dropping alone cannot support the number of resolutions in our experimental setup.

The resolution arrangement for this group of experiments is shown in Figure 3-11. 96×64 is the base layer resolution for the two scalable encoding schemes; 192×168 is the first enhancement layer resolution; and so on. There are six layers altogether for *The Italian Job* and 11 layers for *Street Corner*. For re-encoding, the video is first encoded at the highest resolution (768×512 for *The Italian Job* and 2880×1920 for *Street Corner*), then decoded, and re-encoded at different resolutions.

3.3.3.1.1.1 Limitations on per-resolution bandwidth efficiency

Figure 3-12 shows the bandwidth efficiency of different algorithms under the one-encoding architecture. The bandwidth efficiency is represented by PSNR and video bit-rate. Each line in the figure represents an algorithm; each dot on a line represents a resolution as the resolution increases from left to right along the line. The x value of a dot represents the bit-rate and the y value of the dot represents the PSNR. In general, lines with dots in the upper-left area (high PSNR and low bit-rate) represent good algorithms. Results from the one-encoding-per-resolution architecture are presented as references by the “non-scalable” line.

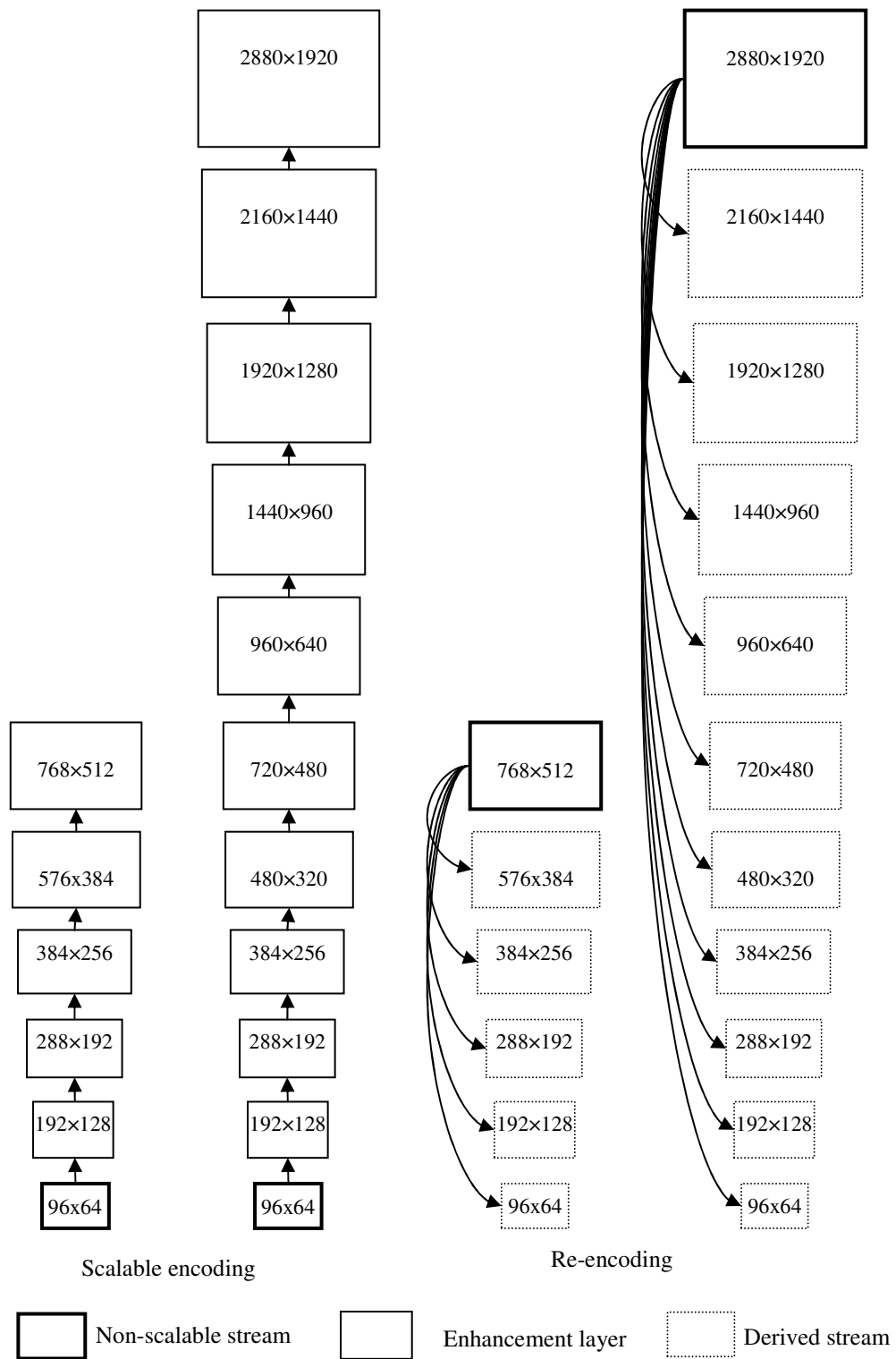
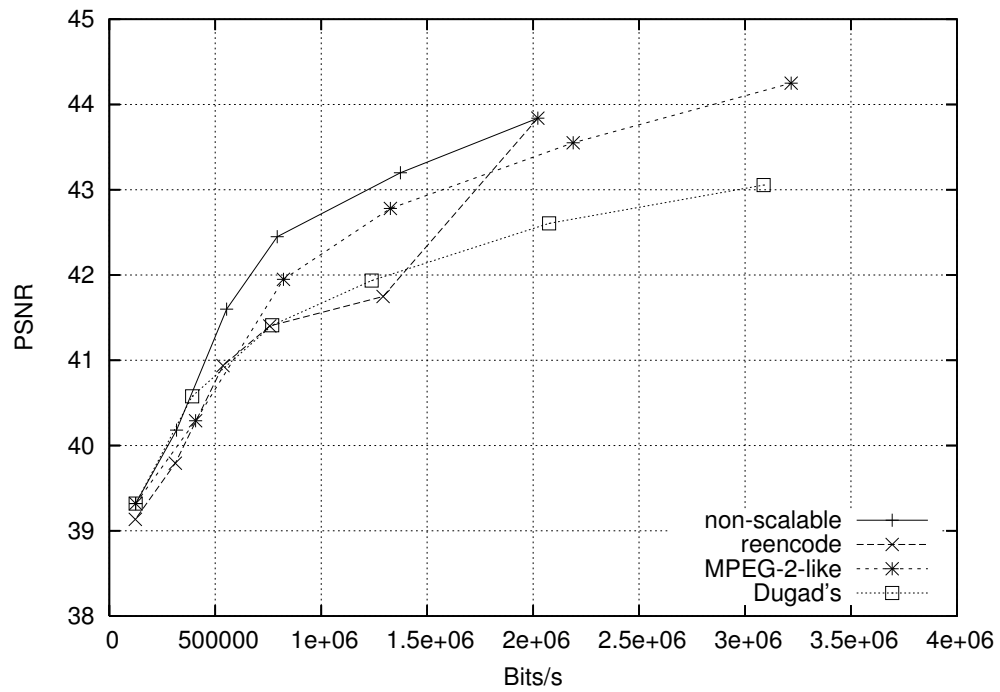
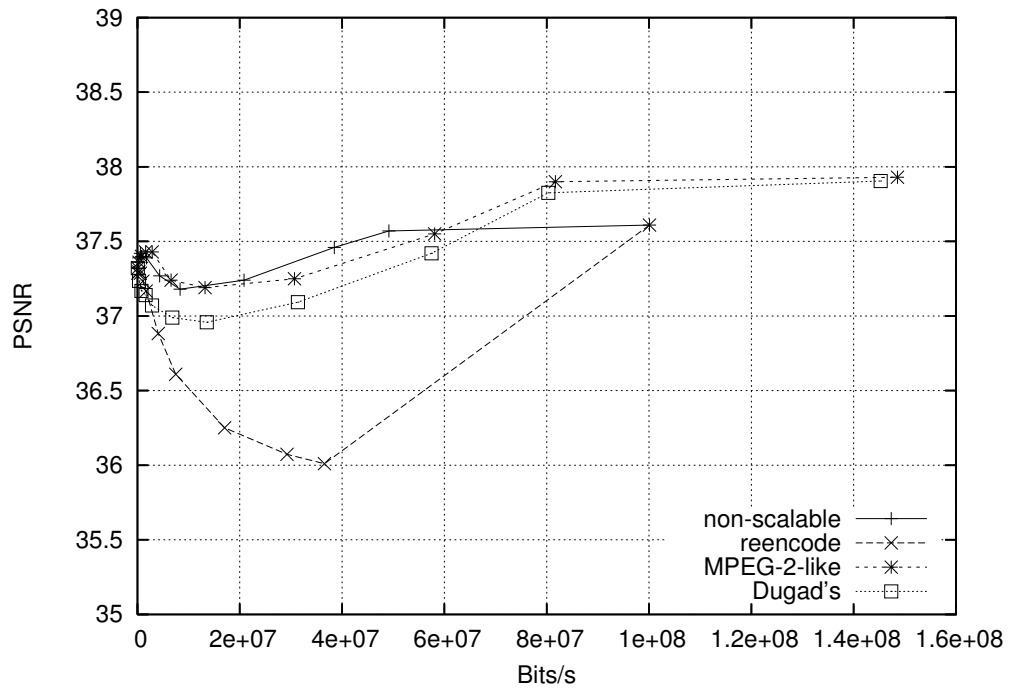


Figure 3-11 The resolution arrangement for mechanisms in the one-encoding-for-all-resolutions architecture.



(a) *The Italian Job*



(b) *Street Corner*

Figure 3-12 Bandwidth Efficiency of Different Algorithms for One-encoding-for-all-resolutions. Each line represents an algorithm. Each dot represents a resolution. All results are from the quantization scale 24. In general, lines with dots in the upper-left area represent good algorithms.

Table 3-3 Compression efficiency of the first enhancement layer and the highest enhancement layer, compared to that of non-scalable encoding.

	<i>The Italian Job</i>				<i>Street Corner</i>			
	<i>192×128</i>		<i>768×512</i>		<i>192×128</i>		<i>2880×1920</i>	
	<i>PSNR</i>	<i>Kbps</i>	<i>PSNR</i>	<i>Kbps</i>	<i>PSNR</i>	<i>Kbps</i>	<i>PSNR</i>	<i>Kbps</i>
Non-scalable encoding	40.18	317	43.84	2022	37.39	260	37.61	100095
MPEG-2 scalable encoding	40.29	407	44.25	3217	37.36	302	37.93	148567
Dugad's scalable encoding	40.58	391	43.05	3088	37.24	304	37.90	145256

For scalable encoding, as the resolution gets higher, the horizontal distance from the “non-scalable” line gets larger, indicating that the stream size of scalable streams grows faster than that of non-scalable streams. To show this trend clearly, we extract the bit-rate and the PSNR for the second lowest resolution (192×128) and the highest resolution (768×512 for *The Italian Job* and 2880×1920 for *Street Corner*) from Figure 3-12 and compare them to those of the non-scalable streams in Table 3-3. The difference in PSNR is insignificant; but the bit-rate overhead compared to non-scalable encoding increases rapidly as the number of layers increases. For the second lowest resolution, a scalably encoded stream consists of one base layer and one enhancement layer; the MPEG-2-like stream of *The Italian Job* is 407 Kbps and is 28% higher than the non-scalable stream at the same resolution; the MPEG-2-like stream of *Street Corner* is 302Kbps and is 16% higher than the non-scalable stream. For the highest resolution of *The Italian Job*, the scalably encoded stream consists of one base layer and five enhancement layers; the MPEG-2-like stream is 3.22Mbps and 59% higher the non-scalable stream. For the highest resolution of *Street Corner*, the

scalable stream has one base layer and ten enhancement layers; the MPEG-2-like stream is 148.567Mbps and 48% higher than the non-scalable stream. The results of Dugad's streams are similar. Even though the increase of overhead is different for the two sequences with different content, simply extending the scalable encoding algorithms to support many resolutions definitely causes overhead accumulation as the number of layers increase thus poor compression efficiency for high resolutions.

For re-encoding, there is a big drop of PSNR for the second highest resolution. This is caused by the artifacts introduced while encoding the highest resolution. A major artifact introduced by a DCT-based compression algorithm is the *blocking artifact*, the discontinuity effect across transform block boundaries. Since the highest resolution is not a multiple of the second highest resolution, block boundaries in the highest resolution get *into* the blocks of the second highest resolution; the blocking artifacts are brought into the blocks too. Thus, when the decoded frames are downsampled to the second high resolution the quality is much worse than downscaling directly from the original highest resolution frames. As the resolution gets smaller, the PSNR of the re-encoded stream gets closer to that of the non-scalable stream at the same resolution because the effect of the artifacts gets smaller since details are lost in low resolutions anyway. Therefore, under the one-encoding architecture, re-encoding cannot efficiently support resolutions close to the encoded resolution.

From Figure 3-12, we also notice that with the same quantization scale, higher resolutions have higher PSNR. We believe the reason is that when an image is

downscaled, gradual variation at the high resolution becomes radical changes within a 4×4 pixel block, which correspond to high-frequency information in the DCT domain that is zeroed out during quantization;

We also notice in Figure 3-12 that, for *The Italian Job*, the coding efficiency of the MPEG-2-like algorithm is much better than that of the Dugad's algorithm especially for high resolutions. We believe that it is because *The Italian Job* is very motion-intensive thus motion estimation is very important for coding efficiency. The enhancement layers in Dugad's algorithm consist of differential signals that are not amiable to motion estimation and motion compensation.

3.3.3.1.1.2 Reducing the number of layers in a scalable encoding

In the previous subsection, we showed that the compression efficiency of scalable encoding degrades rapidly as the number of layers grows and the compression efficiency of the highest resolution is especially poor. In this subsection, we are interested in whether we can improve the compression efficiency of the highest resolution in the one-encoding architecture by reducing the number of layers between the lowest resolution and the highest resolution. The resolutions between layers are derived by dropping DCT coefficients of an enhancement layer, transmitting the partial enhancement layer and all layers below, decoding and downscaling at the client side. This is the mixed scheme that we proposed in section 3.2.1. The base layer resolution, the enhancement layer resolutions, and the resolutions derived from each enhancement layer are shown in Figure 3-13. There are three layers in *The Italian Job* and four layers in *Street Corner*.

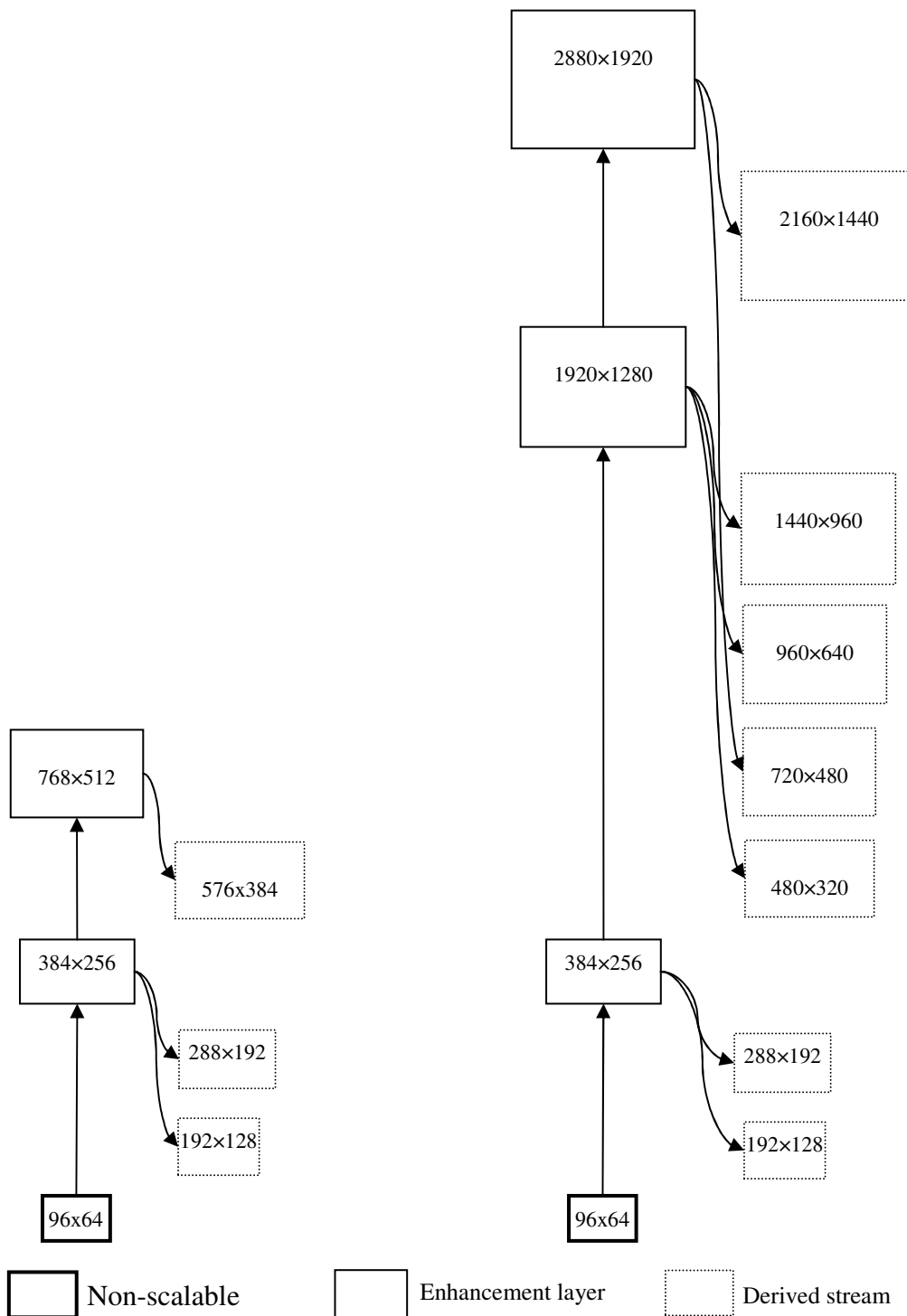
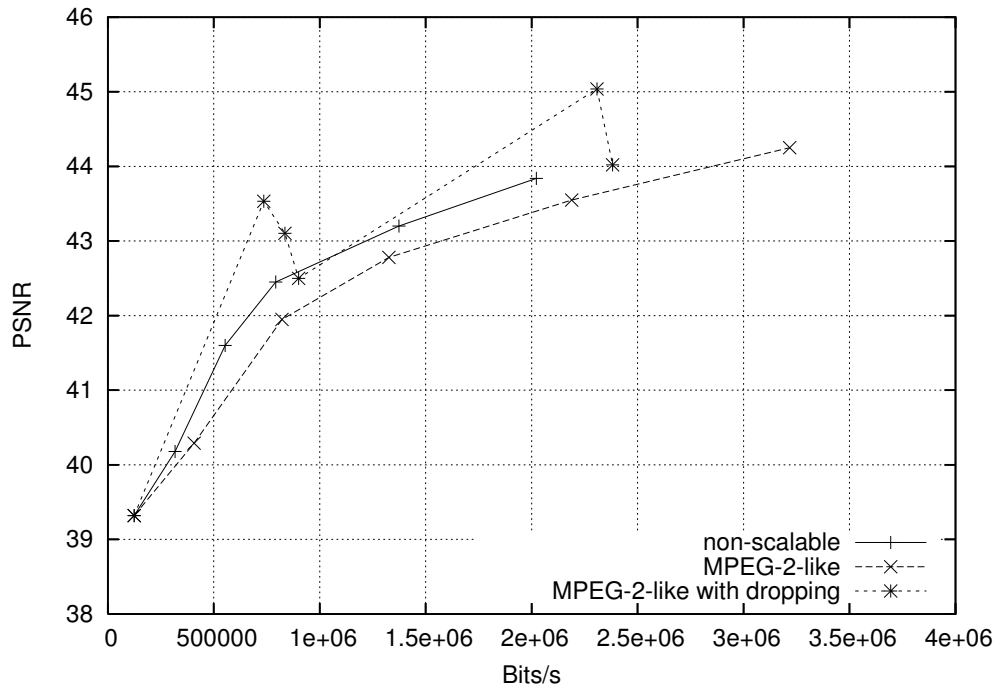


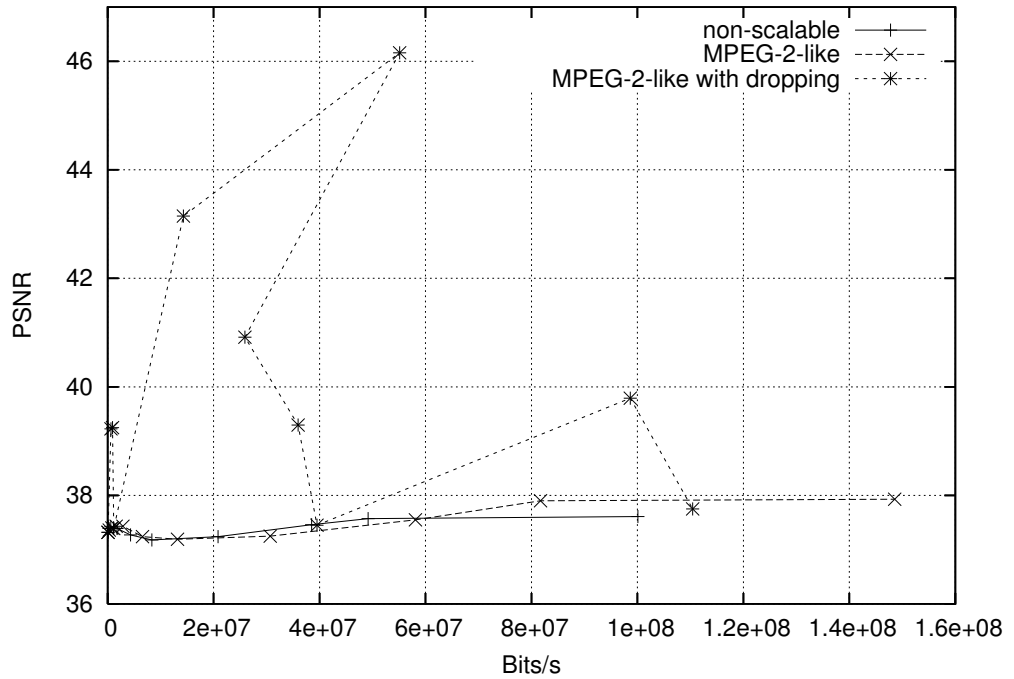
Figure 3-13 The resolution arrangement for the combination of scalable encoding and DCT-coefficient dropping in the one-encoding-for-all-resolutions architecture.

The compression efficiency of the mixed scheme for all resolutions is shown in Figure 3-14 and Figure 3-15, for the MPEG-2-like scheme and the Dugad's scheme respectively. The results for pure MPEG-2-like scalable encoding and pure Dugad's scalable encoding are included for comparison. In both Figure 3-14 and Figure 3-15, the compression efficiency of the highest resolution (the rightmost dot on each line) of the mixed algorithms is improved compared to pure scalable encoding. The compression efficiency of the highest resolution is one of our major concerns and is listed in Table 3-4. For *The Italian Job*, the MPEG-2-like stream with three layers is 2.38 Mbps, which is 26% smaller than the MPEG-2-like stream with six layers in the one-encoding architecture; and the overhead compared to a non-scalable stream is reduced from 59% to 18%. Its PSNR is close to the six-layer MPEG-2-like stream therefore the coding efficiency for the highest resolution is improved when there are fewer layers in a stream. For the *Street Corner*, the MPEG-2-like stream with four layers is 110.4Mbps and is 25% smaller than the MPEG-2-like stream with 11 layers in the one-encoding architecture while the PSNR is similar. Reducing the number of layers, even though the distance between the base resolution and the highest resolution is still the same, improves the coding efficiency for the highest resolution.

In Figure 3-14(b) and Figure 3-15(b), the lines representing the mixed scheme are non-monotonic in that resolution 720×480 has higher bit-rate than resolution 960×640 and resolution 1440×960. This is due to that the decrease in stream size through dropping high-frequency coefficients is very limited, as we show in Figure 3-4, because some of the dropped coefficients are already zeros. Therefore, the number of

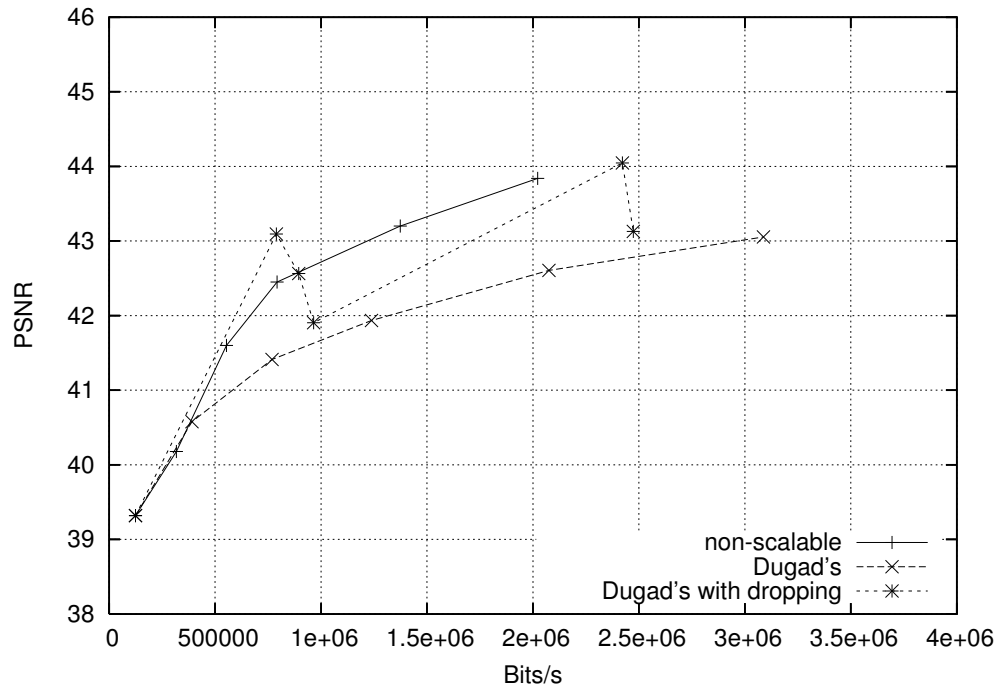


(a) *The Italian Job*

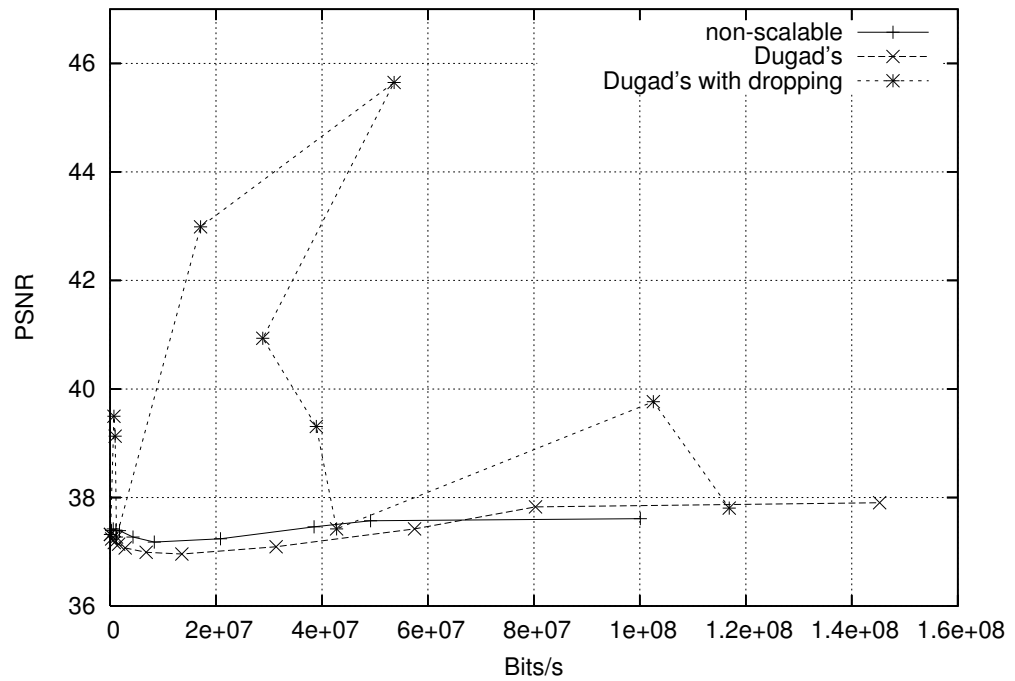


(b) *Street Corner*

Figure 3-14 Comparison of the MPEG-2-like schemes with six layers and three layers combined with DCT-coefficient dropping. Each line represents a configuration. Each dot represents a resolution. All results are from the quantization scale 24. In general, lines with dots in the upper-left area represent good configurations.



(a) *The Italian Job*



(b) *Street Corner*

Figure 3-15 Comparison of the Dugad's scheme with six layers and three layers combined with DCT-coefficient dropping. Each line represents a configuration. Each dot represents a resolution. All results are from the quantization scale 24. In general, lines that are high and with dots close to the Y-axis represent good configurations.

Table 3-4 Compression efficiency of the highest resolution (768×512 for the Italian job and 2880×1290 for street corner)

	<i>The Italian Job</i>					<i>Street Corner</i>				
	Non-scalable	MPEG-2-like	MPEG-2-like + dropping	Dugad's	Dugad's + dropping	Non-scalable-	MPEG-2-like	MPEG-2-like + dropping	Dugad's	Dugad's + dropping
PSNR	43.84	44.25	44.02	43.05	43.12	37.61	37.93	37.75	37.9	37.80
Bit-rate (Mbps)	2.022	3.217	2.38	3.088	2.47	100.1	148.6	110.4	145.3	116.9

macroblocks is the determining factor of the stream size. Resolution 720×480 is generated from 2880×1920 as shown in Figure 3-13 and has more macroblocks than the 960×640 stream and the 1440×960 stream, which are generated from the 1920×1280 stream.

Since the decrease in stream size through dropping high-frequency coefficients is very limited, the resolutions generated by dropping coefficients have large stream sizes (and high PSNRs) compared to non-scalable streams and scalable streams because they are derived from a higher resolution and have more macroblocks encoded. To understand how large the derived streams are, we compare the bit-rate of the derived streams with that of the scalable streams at the same resolution in Table 3-5. We also list how many coefficients are kept out of the 4×4 DCT coefficient matrix for a derived resolution. Table 3-5 shows that when 3×3 coefficients are kept, the derived stream is no more than 35% larger than the scalable stream; when 2×2 coefficients are kept, the derived stream is about one time larger than the scalable stream; and when only one DC coefficient is kept, the derived stream is several times

Table 3-5 Bit-rates of derived resolutions

	Resolutions	Coefficients Kept	Bit-rate (Mbps)			
			MPEG-2-like	MPEG-2-like + dropping	Dugad's	Dugad's + dropping
<i>The Italian Job</i>	192x128	2x2	0.41	0.74	0.39	0.79
	288x192	3x3	0.82	0.84	0.77	0.89
	576x384	3x3	2.19	2.38	2.07	2.42
<i>Street Corner</i>	192x128	2x2	0.30	0.63	0.30	0.73
	288x192	3x3	0.78	0.90	0.78	0.96
	480x320	1x1	2.88	14.30	2.85	17.07
	720x480	1x1	6.57	55.10	6.82	53.60
	960x640	2x2	13.19	25.90	13.54	28.81
	1440x960	3x3	30.69	35.96	31.34	38.93
	2160x1440	3x3	81.70	110.43	80.29	102.54

larger than the scalable stream. For example, for the MPEG-2-like scheme and the *Street Corner* sequence, at resolution 1440x960, the derived stream is 35.96Mbps, which is 17% larger than the scalable stream; at resolution 960x640, the derived stream is 25.90Mbps and about twice the size of the scalable stream; at resolution 720x480, the derived stream is 55.10Mbps and 8.38 times of the scalable stream. We believe that it is acceptable to derive one resolution between layers through dropping the right-most column and the bottom row in the 4x4 DCT coefficient matrix; dropping more than that will cause the stream too big for the derived resolution.

3.3.3.1.3 Improvement in overall bandwidth efficiency

Scalable encoding, at the expense of coding efficiency for each resolution, can support all resolutions in one stream—the stream for the highest resolution—through dropping layers. Therefore, scalable encoding is efficient in supporting multiple

Table 3-6 Overall bandwidth Efficiencies for mechanisms in the one-encoding-for-all-resolutions architecture. Bit-rates are in mega bits per seconds. The bandwidth efficiency for non-scalable encoding is listed as a reference.

	<i>The Italian Job</i>				<i>Street Corner</i>			
	Non-scalable encoding	MPEG-2-like	Dugad's	Re-encode	Non-scalable encoding	MPEG-2-like	Dugad's	Re-encode
Average PSNR for all resolutions	41.765	42.02	41.48	41.14	37.39	37.45	37.28	36.88
The worst PSNR of all resolution	39.32	39.32	39.32	39.13	37.18	37.19	37.09	36.01
Overall bit-rates for all resolutions (Mbps)	5.18	3.22	3.09	5.04	225.0	148.6	145.3	198.1

resolutions at the same time because different resolutions can share data (assuming multicasting is used). The bandwidth requirement for all resolutions is shown in the last row in Table 3-6. Using the MPEG-2 like algorithm, it requires 37.8% less bandwidth for *The Italian Job* and 34.0% less for the *Street Corner* than sending a non-scalable stream for every resolution. The average PSNR and the worst PSNR of scalable encoding are close to those of non-scalable encoding.

Re-encoding also requires less bandwidth than non-scalable encoding but the saving is not as significant as scalable encoding and the PSNR is also a little lower than that of scalable encoding.

Not shown in Table 3-6 is the overall bandwidth for the mixed scheme because it depends on whether the intermediate nodes can perform coefficient dropping. If the intermediate nodes can do coefficient dropping, then all resolutions can share one stream and require less overall bandwidth than scalable encoding because this stream of the mixed scheme has less layers. If the intermediate nodes cannot do coefficient dropping, then the server has to do it and sends the original stream and coefficients-dropped streams. Even though there is still bandwidth sharing among scalably encoded resolutions, these streams generated through dropping coefficients are so large (and cannot be shared) that more bandwidth is required to support all resolutions than transmitting all non-scalable streams

3.3.3.1.2 *The hybrid architecture*

In this subsection, we try to improve the per-resolution compression efficiency by increasing the number of encodings and reducing the number of resolutions supported by each encoding. We have used two encodings for *The Italian Job* and three encodings for *Street Corner*. The resolutions included in each encoding and derived resolutions from an encoding are shown in Figure 3-16.

We first discuss the results for multiple scalable encoding and then the results for multiple non-scalable encoding.

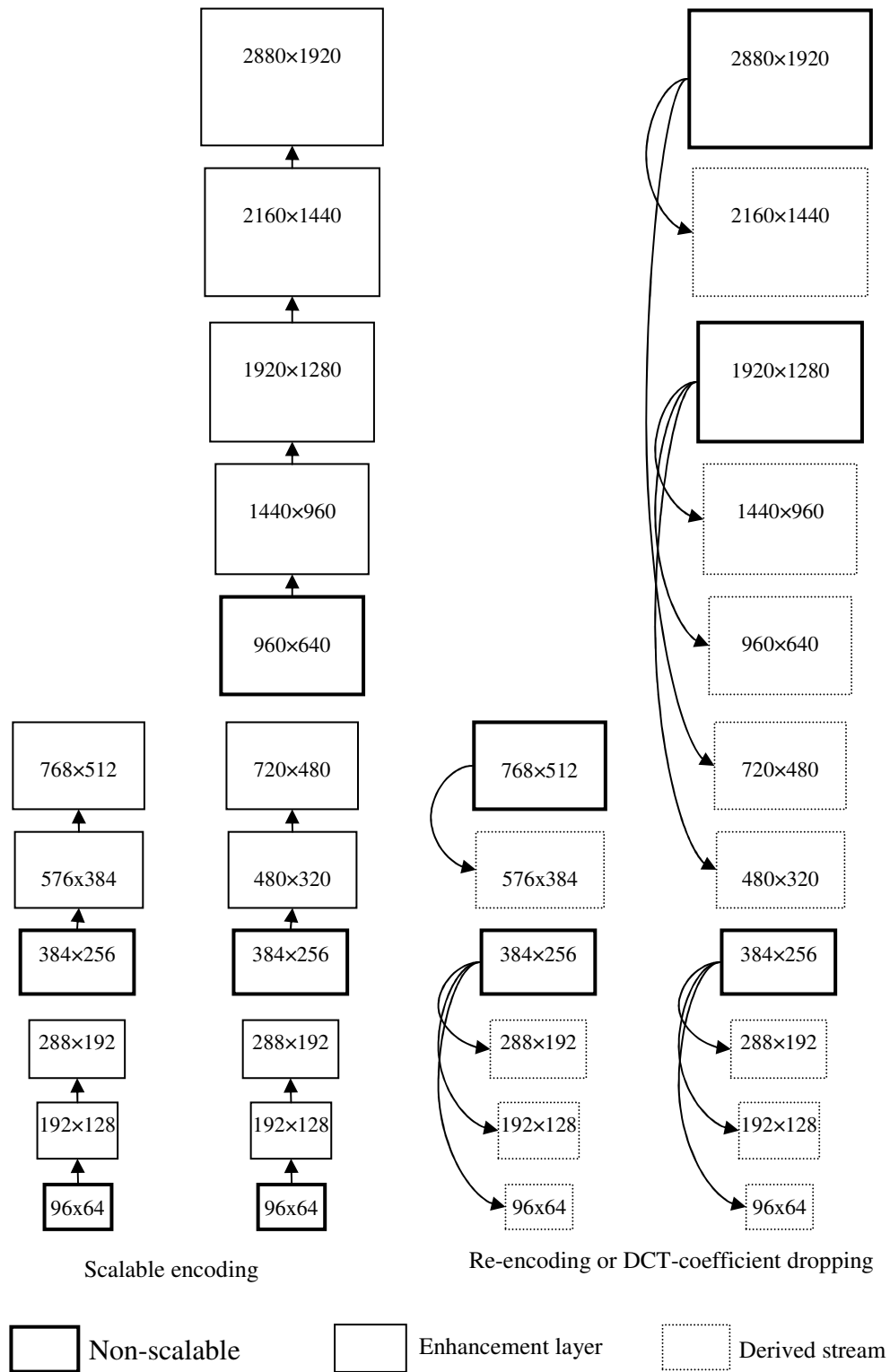
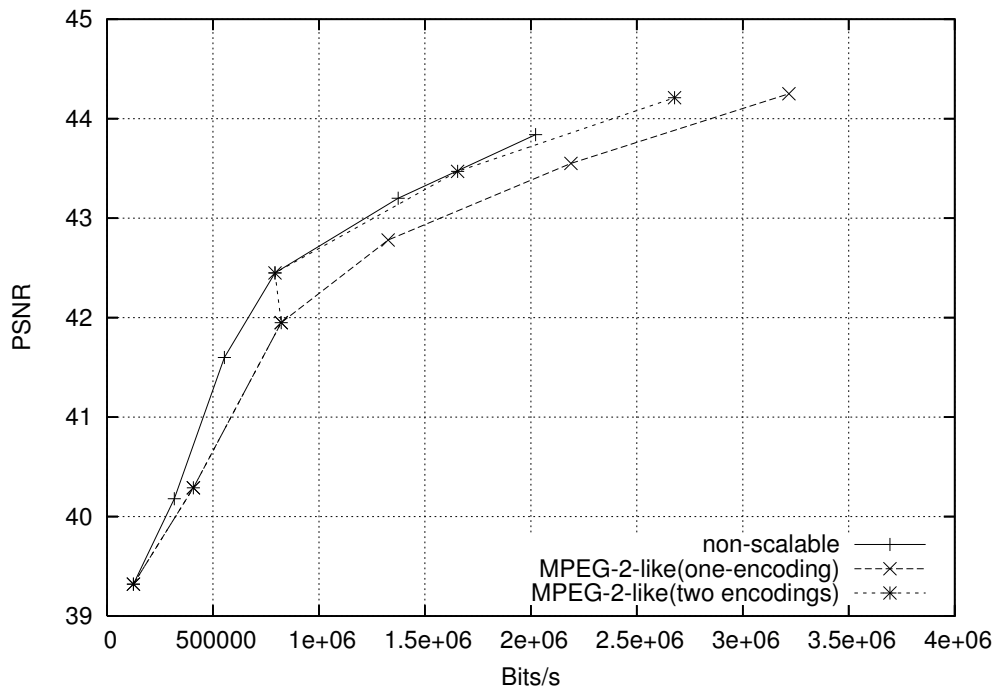


Figure 3-16 The resolution arrangement for configurations in the hybrid architectures.

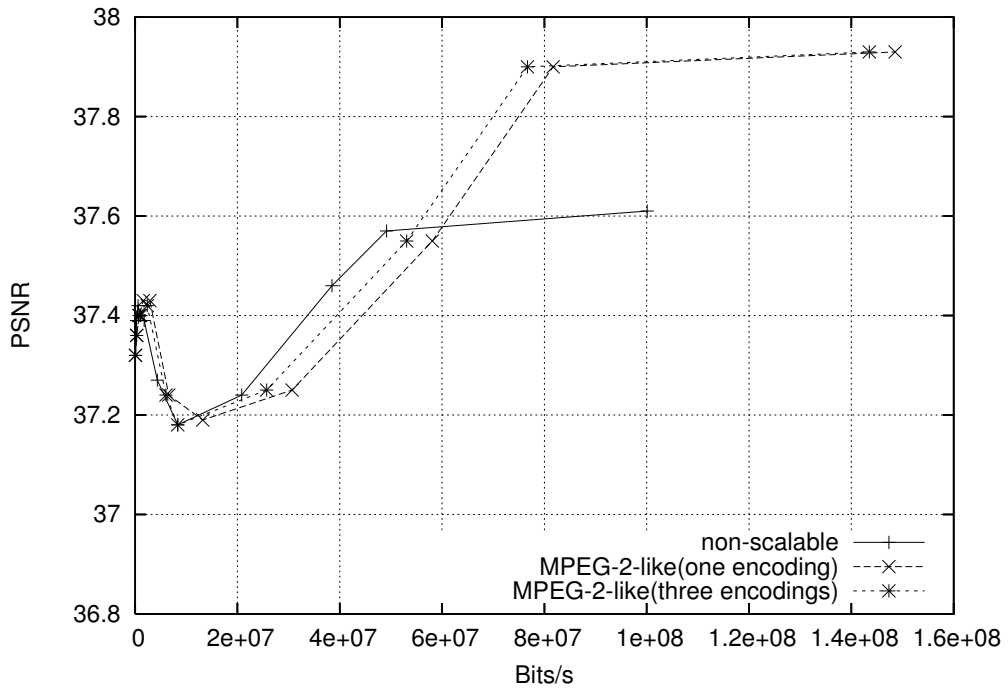
3.3.3.1.2.1 Multiple scalable encodings

In Figure 3-17 and Figure 3-18, we show the compression efficiency at all resolutions for multiple MPEG-2-like scalable encodings and multiple Dugad's scalable encodings, respectively. The compression efficiency of the highest resolution is one of our major concerns and is listed in Table 3-7.

For *The Italian Job*, the stream for the highest resolution consists of one base layer and two enhancement layers. The MPEG-2-like stream is 2.67 Mbps, which is 17% smaller than the MPEG-2-like stream in the one-encoding architecture and the overhead compared to a non-scalable stream is reduced from 59% to 32%. The PSNRs are similar for these two streams therefore the coding efficiency is improved when there are fewer layers in a stream. For the *Street Corner* video, the stream for the highest resolution consists of one base layer and four enhancement layers. The MPEG-2-like stream is 143.5Mbps and is 3.4% smaller than the stream in the one-encoding architecture. The improvement is not as significant as that for *The Italian Job* and we believe that this is due to the five layers in the stream that have already accumulated large overhead. If we look at resolution 1440×960, the first enhancement layer in the five-layer stream, the stream size decreases from 30.67Mbps in the one-encoding architecture to 25.71Mbps in the three-encoding architecture, a 16.17% reduction for the same PSNR 37.25dB. For Dugad's algorithm, the results are similar.

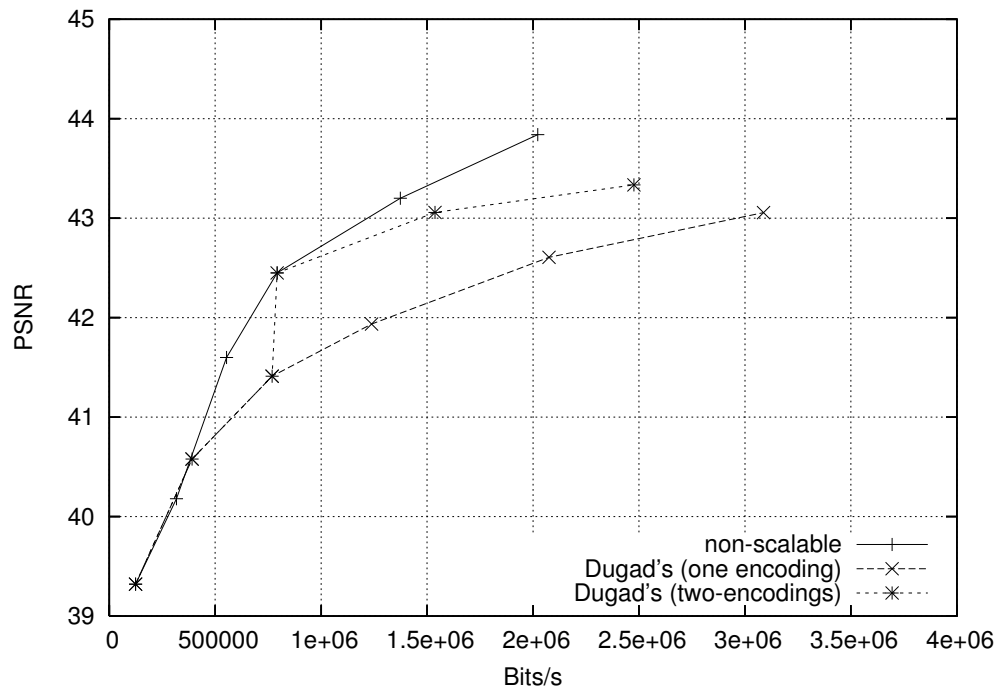


(a) *The Italian Job*

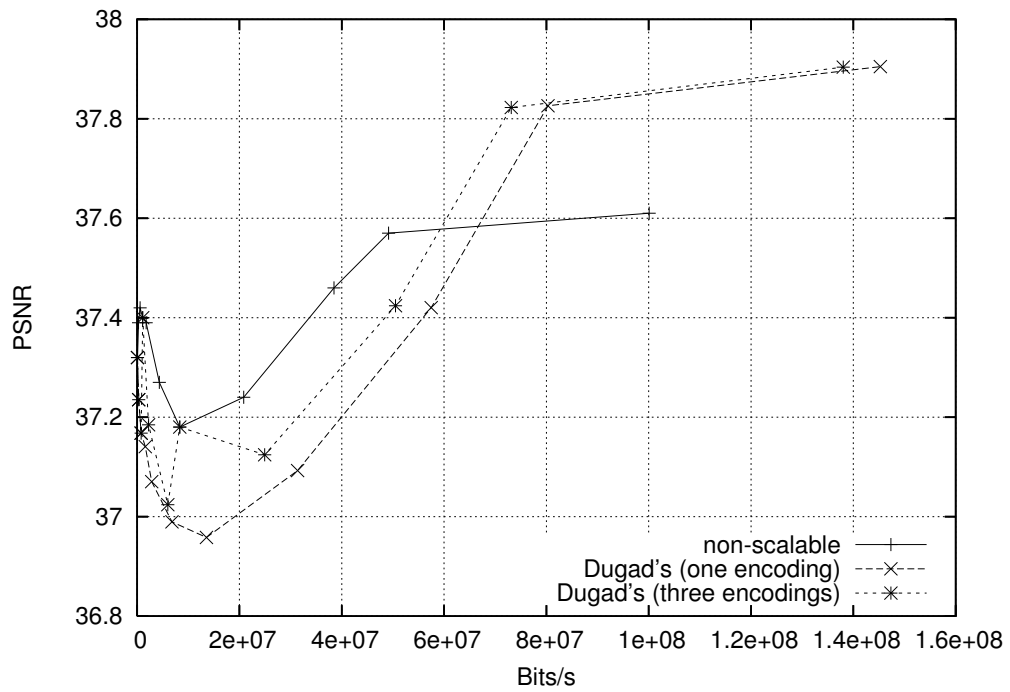


(b) *Street Corner*

Figure 3-17 Comparison of MPEG-2-like schemes in different configurations: one encoding or multiple encodings. Each line represents a configuration. Each dot represents a resolution. All results are from the quantization scale 24.



(a) *The Italian Job*



(b) *Street Corner*

Figure 3-18 Comparison of Dugad's schemes in different configurations: one encoding or multiple encodings. Each line represents a configuration. Each dot represents a resolution. All results are from the quantization scale 24.

Table 3-7 Comparison of compression efficiency for the highest resolution in scalable encodings with different configurations. The PSNR and the stream size for non-scalable encoding are listed as references.

	<i>The Italian Job (768×512)</i>				<i>Street Corner(2880×1920)</i>			
	One encoding		Two encodings		One encoding		Three encodings	
	PSNR	Mbps	PSNR	Mbps	PSNR	Mbps	PSNR	Mbps
Non-scalable encoding	43.84	2.02	43.84	2.02	37.61	100.1	37.61	100.1
MPEG-2-like	44.25	3.22	44.21	2.67	37.93	148.6	37.93	143.5
Dugads'	43.05	3.09	43.33	2.48	37.90	145.3	37.90	138.0

In summary, reducing the number of layers (by adding one or two encodings) in one encoding can improve the coding efficiency of the highest resolution and retain good compression efficiency of other resolutions. The improvement is more significant when there are less than five layers in one encoding.

The overall bandwidth efficiency of multiple scalable encodings is very close to that of the one-encoding architecture and much better than that of the one-encoding-per-resolution architecture as shown in Table 3-8 because the compression efficiency for high resolutions is improved. In the hybrid architecture, we have to stream more than one encoding to support all resolutions at the same time. The total number of layers is the same regardless of the number of encodings since there is always one layer corresponding to one resolution; the number of base layers, however, increases as the number of encodings increases. For example, for *The Italian Job*, the resolution 384×256 corresponds to an enhancement layer in the one-encoding architecture but is a base layer in the two-encoding architecture. Usually a base layer is larger than an enhancement layer for the same resolution; therefore, increasing the number of

Table 3-8 Overall bandwidth Efficiency for multiple scalable encodings. Bit-rates are in mega bits per seconds. The bandwidth efficiencies for one-scalable encoding and non-scalable encoding are listed as references.

	<i>The Italian Job</i>					<i>Street Corner</i>				
	Non-scalable	MPEG-2-like	MPEG-2-like (two encodings)	Dugad's	Dugad's (two encodings)	Non-scalable	MPEG-2-like	MPEG-2-like (three encodings)	Dugad's	Dugad's (three encodings)
Average PSNR for all resolutions	41.76	42.02	41.95	41.48	41.69	37.39	37.45	37.45	37.28	37.34
The worst PSNR of all resolution	39.32	39.32	39.32	39.32	39.32	37.18	37.19	37.18	37.09	37.02
Overall bit-rates for all resolutions (Mbps)	5.18	3.22	3.59	3.09	3.24	225.0	148.6	150.3	145.3	144.8

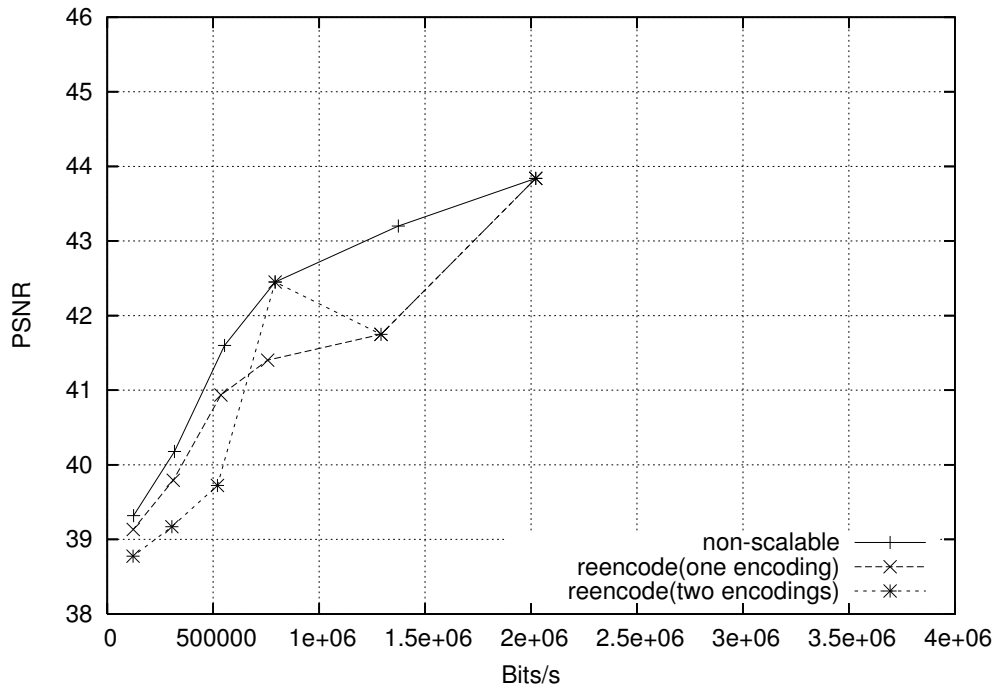
encodings is likely to increase the overall bandwidth requirement because it increases the number of base layers. However, since the coding efficiency for high resolutions is improved, sending one or more encodings does not increase the bandwidth requirement much. This is especially true for the *Street Corner*, which contains many details and reducing the number of layers can greatly increase the coding efficiency. For example, sending the three MPEG-2-like scalable streams requires 150.3Mbps only 1.1% more bandwidth than the one big MPEG-2-like scalable stream (consisting of eleven layers) and still requires 33.2% less than transmitting all non-scalable streams. For Dugad's algorithm, the bandwidth for the three streams is even smaller than that for the one big stream, which means the improved compression efficiency for high resolutions offsets the high bit-rates of the two extra base layers.

3.3.3.1.2.2 Multiple non-scalable encodings

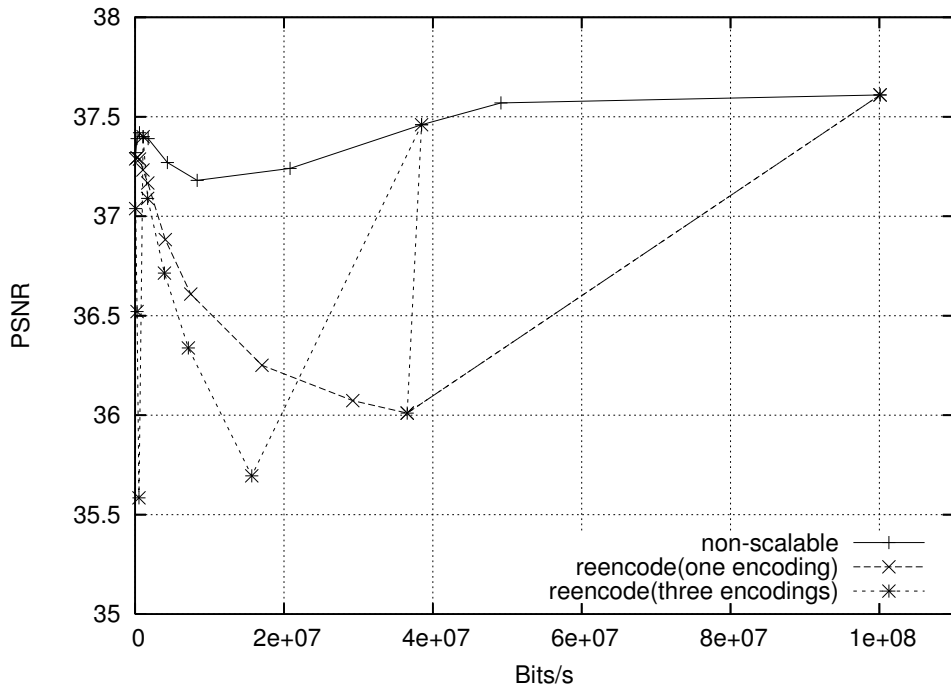
For multiple non-scalable encodings, we have tested two ways to derive resolutions that are not encoded: re-encoding and DCT-coefficient dropping.

In Figure 3-19, we consider multi-encoding combined with re-encoding and compare it with re-encoding in the one-encoding architecture. Adding a new encoding adds a sharp drop for the resolution next to it and decreases the compression efficiency for lower resolutions. This is consistent with our analysis in the previous section that re-encoding is inefficient for those resolutions close to the encoded resolution because of blocking artifacts. For re-encoding, the hybrid architecture performs worse than the one-encoding architecture.

In Figure 3-20, we consider multi-encodings combined with DCT-coefficient dropping. The derived resolutions have very high bit-rates compared to non-scalable encoding because the number of macroblocks is larger in the derived streams. To understand the compression efficiency of these resolutions, we plot the PSNR and bit-rate of resolutions 96×64 , 192×128 , and 288×192 over many quantization scales in Figure 3-21. The compression efficiency for the resolution 288×192 is very close to that of non-scalable encoding when the bandwidth is low and the difference begins to show only when the PSNR is higher than 40dB. The compression efficiencies for the resolutions 192×128 and 96×64 are poor. For example, the PSNR is at least 5dB worse than that of non-scalable encoding at 96×64 . For the same PSNR, for example 40dB, the derived stream is about 280Kbps and is about twice the bit-rate of the non-

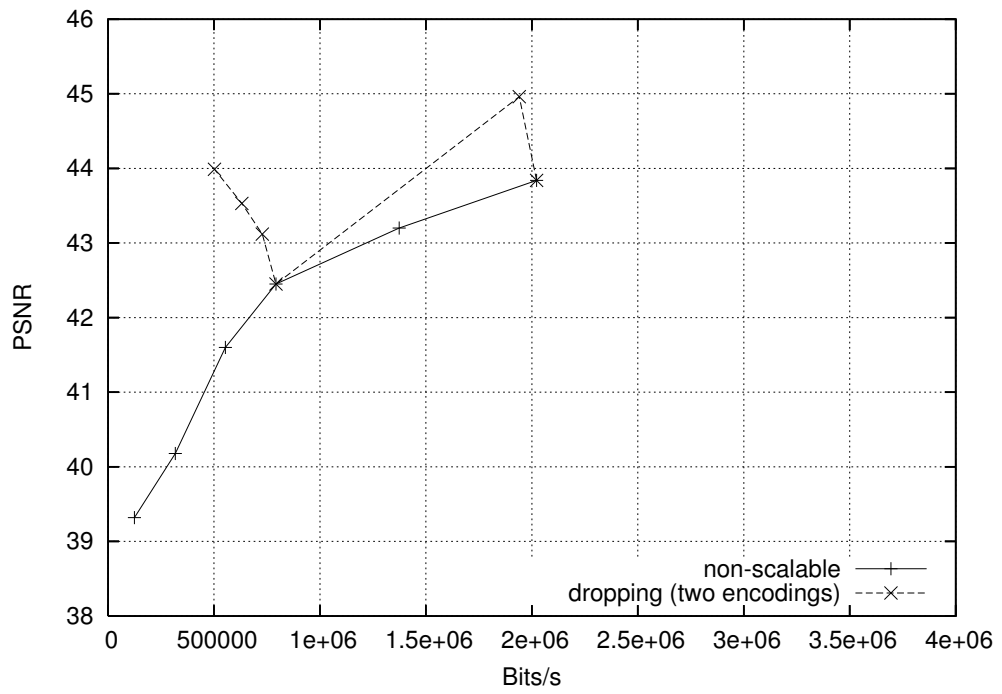


(a) The Italian Job

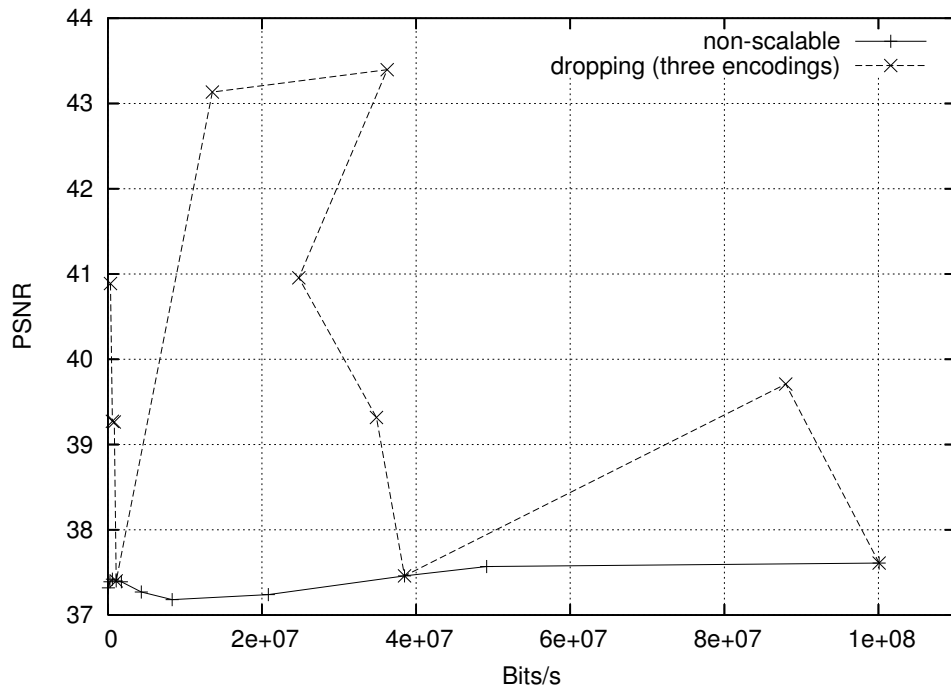


(b) Street Corner

Figure 3-19 Comparison of re-encoding in different configurations: one encoding and multiple encodings. Each line represents a configuration. Each dot represents a resolution. All results are from the quantization scale 24.

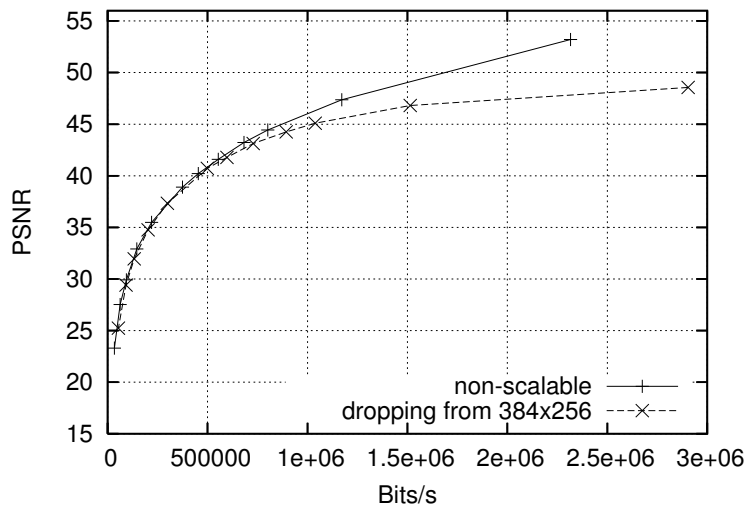


(a) *The Italian Job*

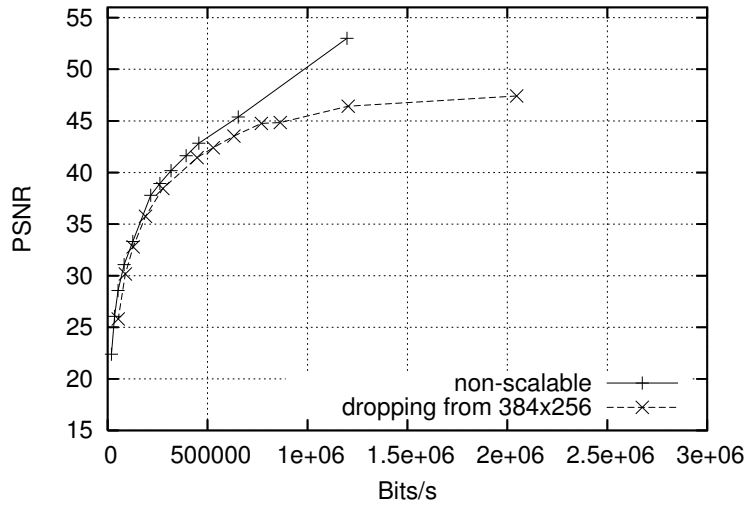


(b) *Street Corner*

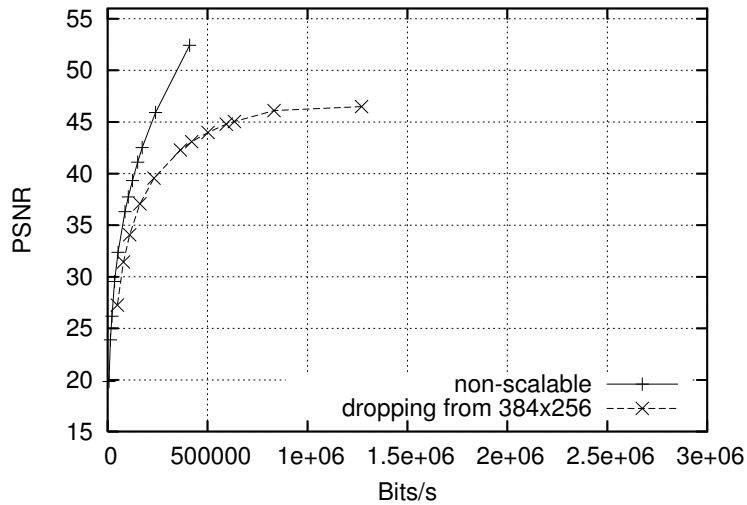
Figure 3-20 Compression efficiency of the combination of multi-encoding and DCT-coefficient dropping.



288x192



192x128



96x64

Figure 3-21 Bandwidth efficiency for single resolutions. The test sequence is *The Italian Job*. Each line represents a configuration and dots on that line represent different quantization scales. DCT-coefficient dropping from multiple encodings is compared with non-scalable encoding and for some resolutions with DCT-coefficient dropping from enhancement layers.

scalable stream. This is consistent with Figure 3-4 in that the compression efficiency gets worse as more coefficients are dropped. Our experiments show that when 3×3 out of 4×4 coefficients are kept after dropping, the compression efficiency is acceptable. Therefore, the number of streams has to be at least half of the number of supported resolutions in this approach.

3.3.3.2 Computational cost

To characterize the computational costs of the various algorithms, we use the number of DCTs, the number of IDCTs, and the number of motion estimations (ME) for encoding a P frame (Y component only) as the measure for computational cost instead of execution time. Since the H.264 codec uses a 4×4 DCT, the number of DCTs/IDCTs is the number of 4×4 blocks in an image (assuming no blocks are skipped). IDCTs in the encoding cycle to generate reference frames are not counted. For each 16×16 block, seven searching modes 16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4 are supported. Therefore, the number of MEs is the number of 16×16 blocks multiplied by seven multiplied by the number of reference frames. We used three temporal reference frames. For the MPEG-2-like scheme, there is one spatial reference frame and three average reference frames so there are seven reference frames in total.

Table 3-9 and Table 3-10 show the computational cost for approaches under the one-encoding architecture, for *The Italian Job* and *Street Corner* respectively. Table 3-11 and Table 3-12 show the computational cost for approaches under the hybrid architecture. From these tables, we conclude that no approaches require

Table 3-9 Computational Cost for mechanisms in the one-encoding-for-all-resolutions architecture and the one-encoding-per-resolution architecture (*The Italian Job*).

		Non-scalable encoding	MPEG-2-like (six layers)	MPEG-2-like (three layers + dropping)	Dugad's (six layers)	Dugad's (three layers + dropping)	Re-encode
Encoding time	No of DCTs	49920	49920	31104	49920	31104	24576
	No of MEs	65520	152208	94584	65520	40824	32256
Streaming time	No of DCTs	0	0	0	0	0	25344
	No of IDCTs	0	0	0	0	0	24576
	No of MEs	0	0	0	0	0	33264

Table 3-10 Computational Cost for mechanisms in the one-encoding-for-all-resolutions architecture and the one-encoding-per-resolution architecture (*Street Corner*).

		Non-scalable encoding	MPEG-2-like (11 layers)	MPEG-2-like (four layers + dropping)	Dugad's (11 layers)	Dugad's (four layers + dropping)	Re-encode
Encoding time	No of DCTs	861120	861120	505728	861120	505728	345600
	No of MEs	1130220	2636508	1548120	1130220	663768	453600
Streaming time	No of DCTs	0	0	0	0	0	515520
	No of IDCTs	0	0	0	0	0	345600
	No of MEs	0	0	0	0	0	676620

Table 3-11 Computational Cost for hybrid architectures: multiple encodings + scalable encoding/transcoding/re-encoding (*The Italian Job*). Non-scalable encoding and one-encoding-for-all-resolutions mechanisms are listed as reference.

		Non-scalable encoding	MPEG-2-like (one encoding)	MPEG-2-like (two encodings)	Dugad's (one encoding)	Dugad's (two encodings)	DCT-coeff dropping	Re-encode (one encoding)	Re-encode (two encodings)
Encoding time	No of DCTs	49920	49920	49920	49920	49920	30720	24576	30720
	No of MEs	65520	152208	141456	65520	65520	40320	32256	40320
Streaming time	No of DCTs	0	0	0	0	0	0	25344	19200
	No of IDCTs	0	0	0	0	0	0	24576	30720
	No of MEs	0	0	0	0	0	0	33264	25200

Table 3-12 Computational Cost for hybrid architectures: multiple encodings + scalable encoding/transcoding/re-encoding (*Street Corner*). Non-scalable encoding and one-encoding-for-all-resolutions mechanisms are listed as reference.

		Non-scalable encoding	MPEG-2-like (one encoding)	MPEG-2-like (two encodings)	Dugad's (one encoding)	Dugad's (two encodings)	DCT-coeff dropping	Re-encoding (one encoding)	Re-encode (two encodings)
Encoding time	No of DCTs	861120	861120	861120	861120	861120	505344	505728	505344
	No of MEs	1130220	2636508	2356284	1130220	1130220	663264	663768	663264
Streaming time	No of DCTs	0	0	0	0	0	0	0	355776
	No of IDCTs	0	0	0	0	0	0	0	505344
	No of MEs	0	0	0	0	0	0	0	46695

significant computation at streaming time except re-encoding. Therefore, computational cost is not a constraining factor for approaches not involving re-encoding.

We also notice other interesting points in the tables. First, the MPEG-2-like scalable encoding scheme requires the most computation at encoding time because of the extra MEs on additional reference frames. Second, re-encoding needs the least computation at encoding time but the computation it saves is spent at streaming time. Thirdly, adding one encoding decreases the encoding-time-computation for the MPEG-2-like scheme because two-encoding means one more base layer that is encoded without additional references. Finally, if we compare the computational cost for *The Italian Job* and *Street Corner*, we can see that resolution is the determining factor for computational cost.

3.3.3.3 Storage cost

Unlike bandwidth, storage is much cheaper. In theory, one can encode video at every resolution that might be used and store it all; in practice, it would be extremely difficult to manage and access the large volume of data; in reality, there is more video produced than stored and even less stored video can be retrieved. Therefore, we still want to lower the volume of data to be stored.

Table 3-13 and Table 3-14 show the storage cost for all mechanisms and architectures. Compared to one-encoding-per-resolution, all other mechanisms can save at least 30% of the storage space. We are especially interested in the storage cost

Table 3-13 Storage Cost for mechanisms in one-encoding-for-all-resolutions architecture, in megabytes and percentage compared to non-scalable encoding.

	Non-scalable encoding	MPEG-2-like	MPEG-2-like + dropping	Dugad's	Dugad's + dropping	Re-encode
<i>The Italian Job</i>	7.72	4.79	3.55	4.60	3.59	3.01
	100%	62.08%	46.00%	59.57%	46.58%	39.00%
<i>Street Corner</i>	112.50	74.28	55.21	72.63	58.43	50.05
	100%	66.03%	49.08%	64.56%	51.94%	44.49%

Table 3-14 Storage Cost for hybrid architectures: multiple encodings + scalable encoding/transcoding/re-encoding, in megabytes and percentage compared to non-scalable encoding. Two encodings for *The Italian Job* and three encodings for *Street Corner*.

	Non-scalable encoding	MPEG-2-like	Dugad's	DCT-coefficient dropping	Re-encode
<i>The Italian Job</i>	7.72	5.21	4.83	4.19	4.19
	100%	67.49%	62.56%	54.27%	54.27%
<i>Street Corner</i>	112.50	75.17	72.40	69.83	69.83
	100%	66.82%	64.36%	62.07%	62.07%

for scalable encoding because they have good overall bandwidth efficiency and low streaming time computational cost. In the one-encoding architecture, the MPEG-2-like scheme saves 37.92% storage space compared to the one-encoding-per-resolution architecture for *The Italian Job* and 33.97% for *Street Corner*. In the hybrid architecture, the MPEG-2-like scheme saves 32.51% for *The Italian Job* and 33.18% for *Street Corner*. Adding one or two encodings does not increase storage cost much for scalable encoding especially for *Street Corner* because the compression is more efficient at high resolutions when there are more than one encoding. The results for Dugad's scheme are similar.

3.3.4 Supporting Multi-resolution Video

In the previous sections, we tried to determine the trade-offs in supporting multi-resolution video through a large number of experiments on real video. However, there are still many open questions. For example, in our experiments, we assume that each resolution is equally important; however, if 80% of users are watching the video at resolution 720×480, how much priority should we give to that resolution?

Indeed, designing a multi-resolution video system poses more questions than those can be answered by our experimental results. In this section, we first try to formulate the problem of designing a multi-resolution video system. Then we describe the guidelines we draw from our experimental results to help the design of such as systems.

The design of a multi-resolution video system can be defined as a Lagrangian optimization problem. Lagrangian methods have been widely used to minimize video quality distortion subject to a rate constraint [58][71][80].

At a high level, the design of multi-resolution video is similar to a typical Rate-Distortion (R-D) optimization problem. A typical R-D optimization problem is to find a solution S that minimizes the Lagrangian cost J , where

$$J = D(S) + \lambda R(S)$$

In the above equation, $D(S)$ is the distortion, $R(S)$ is the rate, and λ is the Lagrange multiplier. If the R-D function is known, for a given λ , an optimal solution $S^*(\lambda)$ that

minimizes the Lagrangian cost J for that λ can be found. The Lagrange multiplier λ allows us to make *a trade-off between the distortion and the rate*. For example, if $\lambda=0$, the optimization is biased to the distortion and minimizing J is to minimize the distortion; if $\lambda=\infty$, the optimization is biased to the rate and minimizing J is to minimize the rate. If there is a rate constraint C , the Lagrangian method can be used to find an optimal or near optimal solution subject to the constraint by looking for a λ that $R(S^*(\lambda))$ is equal or close to C .

In the scenario of video coding, all three quantities D , λ , and R tend to be complicated and subject to approximations and compromises. For example, the use of temporal prediction makes optimization decisions on one frame have cascading effects on subsequent frames and these interactive effects are often ignored in practice [71]. For the design of multi-resolution video, we consider three cost factors: bandwidth, computation, and storage, instead of one. Therefore, both λ and R are vectors and the Lagrangian cost function is:

$$J = D(S) + \lambda_1 R_1(S) + \lambda_2 R_2(S) + \lambda_3 R_3(S),$$

where R_1 is bandwidth, R_2 computation, and R_3 storage; and $\lambda=(\lambda_1, \lambda_2, \lambda_3)$ is the vector Lagrange multiplier.

The vector Lagrange multiplier λ allows us not only to make a trade-off between the distortion and the resource consumption but also to make trade-offs among different resources. For example, bandwidth is usually more expensive than storage; this can be reflected in the optimization process by specifying a λ_1 larger than λ_3 .

However, the choice of λ is also subject to $R_i(S^*(\lambda)) \leq C_i$ for $i=1, 2$, and 3 , where C_i is the constraint of the corresponding resource.

The biggest problem for designing multi-resolution video is to establish the right R-D function. If the R-D function is known, searching for an optimal solution for a given λ and searching for a λ whose optimal solution can make the best use of available resources have been studied [9][23][57][61]. However, establishing the R-D function for designing multi-resolution video is extremely difficult because the huge solution space and many application-specific factors. We discuss the solution space and application-specific factors below.

A solution S for our design problem consists of several parts:

$$S = (s_1, s_2, s_3, s_4),$$

where s_1 is the grouping of resolutions, s_2 is the mechanism to support multiple resolutions in one group, s_3 is the codec, and s_4 is the set of encoding parameters.

The above definition of the solution is quite simplified. For example, we assume that each group uses the same mechanism to support multiple resolutions. However, the number of possible solutions is still large, just considering the pages of encoding parameters for the H.264 codec.

The measure of the distortion is highly application-specific. Assuming MSE (Mean Square Error) is used to measure distortion, the distortion of the system depends on the MSE at each resolution:

$$D(S) = \frac{1}{N} \sum_{i=1}^N w_i MSE_i(S)$$

where N is the total number of resolution and w_i is the weighing parameter reflecting the important of resolution i .

The weights of resolutions are application-specific. For example, some applications may give priority to high resolutions and others may give priority to resolutions with the most users. In addition, the MSE for each resolution also depends on the content of the video, which is application-specific and dynamic.

The measure of resource consumption is also application-specific. For example, the bandwidth consumption depends on the underlying transport protocol and the geographic distribution of users; the computation depends on hardware and the operating system.

While capturing R-D characteristics needs further research and is very application-specific, our work has established some general guidelines to quickly discard bad solutions and find promising solutions in the framework of *Bonneville*.

Our experiments focus on exploring s_1 (the grouping of resolutions) and s_2 (the mechanism to support multiple resolutions in one group) in the solution space. Our work shows that one single encoding or a single algorithm is hard to provide fine-grained wide-range multi-resolution video effectively. We have also run similar experiments [35] using the *ffmpeg* MPEG-1 codec [22], which is known for its speed

while the compression efficiency is not optimized. Different codecs results in different results. For example, scalable encoding is more efficient with the H.264 codec; and the 8×8 DCT transform of the *ffmpeg* codec allows a wider working range of coefficient dropping. However, our results all indicate a hybrid architecture can provide a good trade-off in resolution adaptation. In particular, we believe that multiple scalable encodings with less than five layers in each encoding are a good start. DCT-coefficient dropping is preferred to generate one resolution between layers if DCT-coefficient dropping can be done at intermediate nodes.

3.4 Conclusions

In this chapter, we have examined the various trade-offs in supporting wide-range, fine-grained multi-resolution adaptation. We believe that in the future, video streaming algorithm for both stored and live video are going to have to potentially support both extremely high-resolution video mapped to a large number of display characteristics. In addition, we believe that such systems will also need to support efficient region-of-interest cropping, especially for applications such as telepresence. Our results show that encoding a video stream for every display size results in highly compressed and optimized video streams. The main drawback of this approach is the high computational complexity required to churn out a potentially large number of streams. Scalable encodings and fast-transcoding are useful but cannot support an extremely wide-range of display characteristics. Finally, our results show that adapting a video stream between relatively close resolution requirements makes sense.

CHAPTER 4

STEENS: MULTI-HOP BUFFERING AND ADAPTATION FOR VIDEO COLLECTION IN SENSOR NETWORKS

As video sensor networks become more widely deployed, mechanisms for adaptively transmitting video data within the network are necessary because of their generally large resource requirements compared to their scalar counterparts. In this chapter, we propose *Steens*, a multi-hop buffering and adaptation framework, for adaptively collecting video in sensor networks. We will show that in-network adaptation and collaboration among sensor nodes is necessary to collect the most useful video with minimal wastage of networking bandwidth.

4.1 Introduction

With recent advances in hardware technologies, the construction of massively scalable video sensor networks is becoming possible. Many applications that rely on video sensor networks require *video collection*, in which the video needs to be sent to a central sink (or sinks) for later analysis and processing. Often, there is no direct network connection between a video sensor and the sink in the sensor network. As such, they typically need to rely on other nodes in the network to buffer and forward data on their behalf.

Because image and video data can represent a large burden on the sensor-networking infrastructure, simply passing data towards the sink, as in scalar sensor networks, may result in network congestion and random dropping of video data in the network, which will lead to waste of bandwidth and rapid degradation of video quality. There are techniques [66][79] to deal with network congestion in a scalar sensor network, but they do not provide appropriate buffer management and data adaptation functions that are needed to adaptively transmit video. Video adaptation techniques are needed to manage in-network buffer space and to tailor video according to network conditions.

Adaptive video collection in such sensor networks cannot be addressed by existing video adaptation mechanisms meant for streaming video over the Internet or other IP-style networks. First, existing adaptation mechanisms for video typically assume end-to-end semantics between them, which does not exist for the *multi-hop store-and-forward routes* in most sensor networks. Second, most of the current streaming algorithms use either a one-to-one unicast or a one-to-many multicast delivery mechanism while video collection is typically *many-to-one*. Finally, existing streaming mechanisms have to satisfy a real-time or “just in time” delivery requirement for video streaming and might not be suitable for video collection, in which video can sit in the network for a much longer time. Furthermore, because the video sensor network will typically respond to specific events spaced out over time, the resultant video may not necessarily be continuous over time; rather, the video will consist of a number of short segments representing events.

There exist a couple of video adaptation systems that adapt video over multi-hop routes or for many-to-one flows; but their applications scenarios are different from video collection in a sensor network. A system that adapts video over multiple hops is End System Multicast [11], which is a video conferencing system on application overlays. In this system, the intermediate nodes are also receivers and adaptation is *independently* performed for each hop. In a sensor network, however, the intermediate nodes are also sources not sinks; and the adaptation must be performed between the sources and the sink through multiple hops. An adaptation mechanism that deals with many-to-one video flows is PALS [63], which streams video from multiple senders to one receiver. However, the multiple streams in aggregate make up a single video stream. In the video sensor case, the multiple streams are distinctly different streams. The adaptation challenge for multiple senders is to choose a subset from the senders and assign different parts of the same video for them to send so that the receiver can get a complete copy of the video. Multiple sources generate different video and the adaptation challenge is to collect the most useful video from all sources. In summary, these adaptation technologies cannot address the requirements of video collection in a sensor network even though they seem to have extra features compared to common adaptation technologies.

In this chapter, we propose *Steens*, a multi-hop video buffering and adaptation framework for video collection in sensor networks. In this framework, nodes in a multi-hop route collaboratively participate in video adaptation and delivery of video sensor data. We describe this framework in section 4.2. In section 4.3, we compare

video quality, bandwidth wastage, and bandwidth sharing fairness of different approaches within this framework and compare them with traditional IP-based video adaptation mechanisms through trace-driven simulations.

4.2 Design of a Multi-hop Buffering and Adaptation Mechanism

Adaptation mechanisms need to know the network conditions to make proper adaptation decisions. However, the lack of end-to-end semantics in a sensor network makes it hard for a sensor node to detect network conditions several hops away. Therefore, we propose to adapt video *hop-by-hop* according to the network conditions on directly connected links, instead of conventional *end-to-end* adaptation that depends on end-to-end network conditions. In this section, we describe the design of *Steens*, a multi-hop buffering and adaptation framework for video collection in sensor networks. We discuss the basic adaptation mechanisms we choose to use in *Steens* and possible ways to compose them to construct a system-wide collaborative adaptation mechanism that can effectively collect data in a multi-hop network and allow bandwidth fair sharing among multiple sources.

For the purposes of this dissertation, we assume that network setup protocols exist to construct and maintain the network topology. We also assume that data loss is caused by congestion only, i.e., links between any two nodes are reliable through link layer retransmission and the adaptation mechanisms have control over data dropping.

4.2.1 Basic Tailoring Mechanism

For *Steens* we will use scalable encoding as the basic tailoring mechanism. There are a number of reasons for this. First, tailoring scalably-encoded video allows video to be adapted through dropping of data, leading to simplified computational requirements for adaptation. Re-encoding or transcoding is too computationally intensive and does not scale because intermediate nodes close to the sink have to tailor multiple video streams from different sources. Second, it does not require transmission of extra data to perform tailoring on intermediate nodes. Multi-encoding requires transmitting multiple streams to intermediate nodes, which can be prohibitively expensive. Finally, most video encoding schemes support scalability to some degree and can be used in *Steens*. For example, even “non-scalable” encodings can support temporal scalability by dropping frames.

4.2.2 Basic Adaptation Mechanism

In order to adapt the video within the network, we will use a priority-based buffering and adaptation mechanism on individual nodes as shown in Figure 4-1. At any time, high priority data (priority zero is the highest and priority three the lowest) in a buffer are sent before low priority data. This is similar to the window-based priority dropping mechanisms in [20][44] except that the “window” here is as large as the buffer. There are a number of advantages of the choice of scalable encoding with priority-based data dropping for adaptation. First, it does not require much computation, which will not burden resource-constrained sensor nodes. Second, it effectively uses buffering for adaptation and can potentially make good adaptation

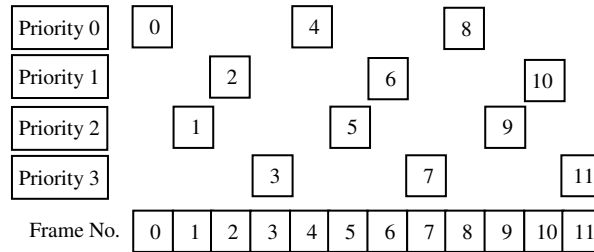


Figure 4-1. The basic adaptation mechanism and a simple prioritization mechanism. At any time, high priority data are sent before low priority data.

decisions by allowing the decisions to be delayed until more information is available. Finally, it separates application-specific encoding-scheme-dependent prioritization from the general scheduling (sending and dropping) algorithm. The prioritization mechanism shown in Figure 4-1 is very simple and it tries to maintain a smooth frame rate based on the assumption that all frames are independently encoded. More complicated prioritization mechanisms can be easily integrated in without affecting the generality of our discussion on adaptation mechanisms.

4.2.3 Composition

Given our chosen adaptation and tailoring mechanisms, the most important question in the design of *Steens* is how to compose adaptation mechanisms on individual sensor nodes into a whole system to achieve the application's adaptation goal. We identify three components that can change the behavior of the composed system: global prioritization, buffer management, and signaling. The global prioritization component accounts for the relative importance among video sources; the buffer management component allocates buffer space among various sources; and the signaling component exchanges information among neighbor nodes to help manage buffers and make adaptation decisions. In the remainder of this section, we

briefly describe the three components and their impact on video quality, bandwidth wastage, and bandwidth sharing in more detail.

4.2.3.1 Global prioritization

In Section 4.2.3, we assume a prioritization mechanism that maps the utilities of data to priorities within one stream. When we combine adaptation mechanisms on individual nodes together, however, prioritization also needs to consider the relative importance of different video sources. For example, data from a camera at a security door are likely more important than data from a camera at an office.

In this thesis, we assume a global prioritization component to map local priorities to global priorities to reflect the importance of video sources. Because such networks are collaborative rather than combative, we believe that this assumption is reasonable. Example global mappings are shown in Figure 4-2. Figure 4-2(a) shows two video sources that are equally important and their local priorities are mapped to the same local priorities. In Figure 4-2(b), *src1* is more important than *src2* and all its data should be sent before data from *src2*. Figure 4-2(c) shows that *src2* is less important than *src1* in general but its highest priority data are as important as those from *src1*. For example, *src2* is the camera at the office, which is usually not as important as camera *src1* at the security door; but frames from *src2* capturing things being moved out of the office are of great importance. The mapping is application-specific and may change over time; yet the adaptation mechanism shown in Figure 4-1 still works because it works on the general notion of priorities.

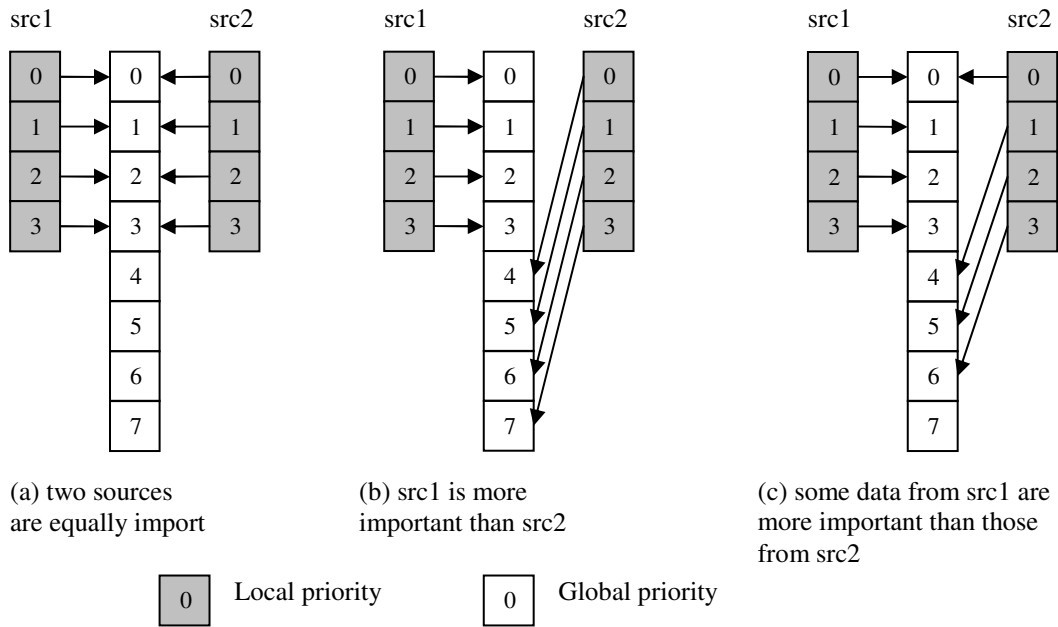


Figure 4-2 Examples of global prioritization.

Global prioritization helps enforce bandwidth sharing among multiple video sources because it reflects the importance of video sources. The importance of video sources defines the goal of bandwidth sharing. In Figure 4-2(a), the two sources should share the bandwidth equally. In Figure 4-2(b), if there is only bandwidth for half of the data, then all bandwidth should be taken by *src1*. The bandwidth-sharing goal is achieved at the granularity of priorities as long as the adaptation mechanism can get as much high priority data to the sink as possible. If the adaptation mechanism can achieve the fair-sharing goal in Figure 4-2(a), it can achieve the biased goal in Figure 4-2(b). In this thesis, we will assume that all video sources are equally important and the mapping in Figure 4-2(a) is used. We will focus on equal sharing

for the rest of this chapter and our discussion can be easily generalized to unequal sharing specified by prioritization.

4.2.3.2 Buffer management

The buffer space on a sensor node is shared by all sensor nodes using it to get data to the sink. For a single source, the buffer space on intermediate nodes can be used for video adaptation; if intermediate nodes can carry out the source's adaptation policy, the effective buffer space for the source is extended and better video quality can be achieved. For multiple sources, how the buffer space is shared affects bandwidth sharing.

There are two primary ways to manage buffers shared by multiple sources. They can either share a single buffer in a first-come-first-serve manner or explicitly partition the buffer amongst the sources. Partitioning can prevent a node from using more resources than its fair-share. However, underutilization of buffer space may happen if the partitions are not updated with network topology changes; for example, if a source nodes becomes isolated, the partition reserved for it becomes empty and cannot be used by another sensor whose partition is overflowing.

4.2.3.3 Signaling

Although adapting video in the network can help realize an application's adaptation policy, there are two problems if they make adaptation decisions using only local information. One is *priority inversion*, a phenomenon in which data dropped by a node might have a higher priority than data kept on other nodes and eventually make

their way to the sink. The other is unnecessary bandwidth wastage, caused by data being sent into the network but dropped somewhere along the way to the sink.

Exchanging information among sensor nodes can help make more globally optimal adaptation decisions at the expense of signaling messages. We propose two signaling protocols for information exchange in this chapter and study their effects on reducing priority inversion and bandwidth wastage as well as the cost of signaling messages.

The first signaling protocol is similar to the ECN (Explicit Congestion Notification) mechanism [62] in the Internet to prevent congestion. ECN sends a “buffer full” message when a high watermark is reached or a “buffer not full” message when a low watermark is reached to upstream nodes toward the source. Nodes receiving the “buffer full” message will stop sending data to that node. A sample message sequence is shown in Figure 4-3(a). We believe that this signaling protocol can reduce bandwidth waste because it helps prevent data that will be dropped at a congestion node from being sent. It has other advantages as well. First, it helps use in-network buffer space effectively by trying to delay dropping until all buffers along the route from the congested node back to the source are full. Since buffers on nodes close to the source usually are not shared by as many streams as buffers on nodes close to the sink, they can greatly increase in-network storage capacity while the network is disconnected. Second, it might reduce priority inversion because it pushes dropping back to the source and reduces dropping at multiple nodes. Third, the buffer full level

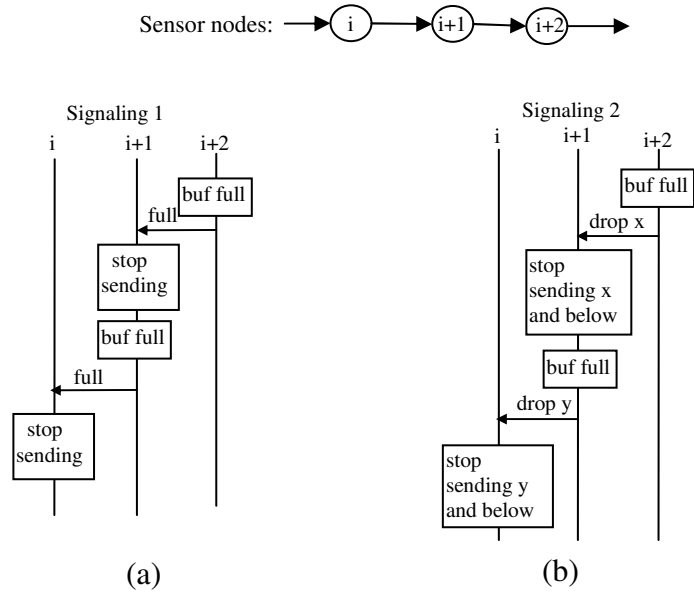


Figure 4-3. Signaling protocols for collaboration among nodes.

is general information so it can be used in a multi-modal sensor network to reduce bandwidth wastage for all types of data. However, this signaling protocol might cause unfair sharing of resources on a certain node because it biases towards nodes close by, which can potentially occupy the buffer before data from nodes farther away arrive.

The second signaling protocol we propose adjusts the dropping level when a high watermark or a low watermark is reached and sends the dropping level, instead of the “buffer full” message, to upstream nodes towards the source so they will not send data that will ultimately be dropped. A sample sequence of messages of this signaling protocol is shown in Figure 4-3(b). This signaling protocol can save bandwidth wastage too but not as aggressively as the first signaling protocol because it allows high priority data to be sent to a congested node and forces the congested node to drop lower priority data arriving earlier. However, by allowing high priority data to be sent

to the congested node, it uses in-network buffers for more effective adaptation. By moving high priority data towards buffers close to the sink, high priority data from all sources are likely to be sent to the sink before low priority data; therefore, it connects the distributed buffers together as if the adaptation is based on a larger buffer. By using a “larger” buffer for adaptation, it reduces priority inversion further and improves fair sharing.

The purpose of these two signaling protocols is to show the benefits of collaboration among sensor nodes; therefore, their design is much simplified. For example, we use the instantaneous buffer fill level to measure congestion or to determine the dropping level. Systems based on instantaneous information tend to be unstable and history information of the buffer can be used to reduce thrashing in a way similar to how it is used in active queue management in the Internet [5][18].

4.3 Experimentation

In this section, we construct trace-driven simulations to verify the advantages of hop-by-hop adaptation over end-to-end adaptation and explore different design parameters in *Steens*.

4.3.1 Simulation Setup and Metrics

For our simulations, we captured a 3,000-frame trace using a Panoptes video sensor [19]. The resolution of the video is 320×240 pixels and the average frame size is 17,282 bytes. This results in a video stream of approximately 4.14Mbps (at 30 frames per second) for each camera. Figure 4-4 shows the network structures we use

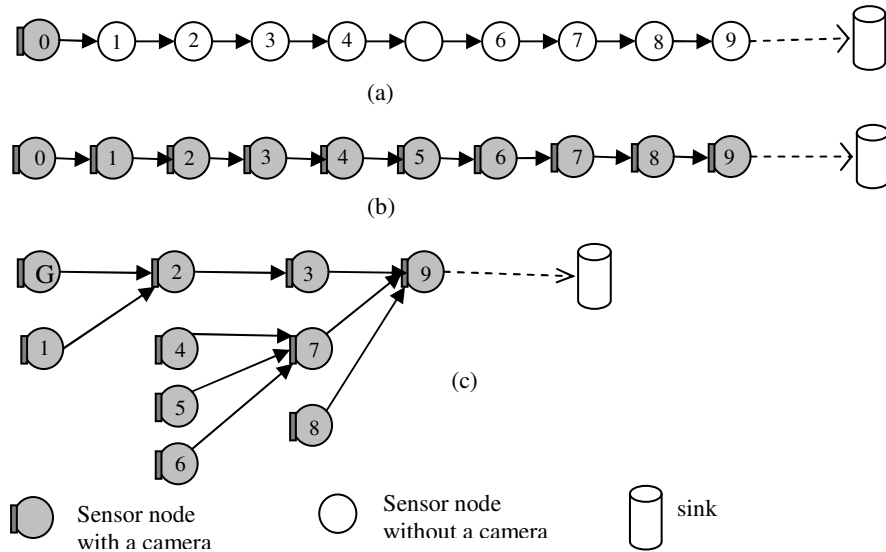


Figure 4-4 The network structures used in the simulations.

in the simulations. The single-video-source structure shown in Figure 4-4(a) will be used when bandwidth sharing is not a concern. Because the last link to the sink is typically shared by the most sensor nodes, we assume that it is the bottleneck link. The results, however, should generalize to any network configuration where the bottleneck is between the source and the sink. We assume that each sensor has 1.5Mega Bytes buffer space. Each simulation run is 100 seconds of simulated time.

The purposes of the simulations are (i) to understand the effectiveness of hop-by-hop adaptation for each video source and (ii) to understand the sharing of bandwidth among multiple resources.

The metrics we use to compare the effectiveness of video adaptation are video quality and wasted bandwidth. Video quality is measured as the throughput and the priority distribution of received frames. Video frame rate smoothness is also used to

compare video quality since the prioritization mechanism we choose aims to deliver a smooth frame rate. There are two types of wasted bandwidth. One is *unconsumed* bandwidth due to buffer underflow, improper scheduling, and so on. The other is bandwidth *consumed* but not contributing to moving data to the sink. In this chapter, we concentrate on the latter because it also wastes energy, a precious resource in sensor networks. There are two sources for consumed but wasted bandwidth: data dropped after leaving their sources and signaling traffic. In our experiments, dropped data are also weighted by the distance from the source because data dropped far away from the source consumes more energy than data dropped closer to the source and we assume transmission over each hop consumes the same energy.

The metric for bandwidth sharing is fairness because we assume that all video sources are equally important as discussed in Section 4.2.4.1. We use the distribution of received frames for each camera to measure bandwidth-sharing fairness.

4.3.2 A Case for Hop-by-hop Adaptation

In this subsection, we show through simulations that hop-by-hop adaptation is better than end-to-end adaptation in terms of video quality. We consider end-to-end adaptation without end-to-end reliability, in which data may be dropped in the network randomly, and end-to-end adaptation with end-to-end reliability. We have implemented a simple end-to-end reliability scheme similar to that in [78], in which a video frame is kept until an acknowledgement is received from the next hop. Figure 4-5 describes the acknowledgement sequences and data loss detection of the reliability scheme.

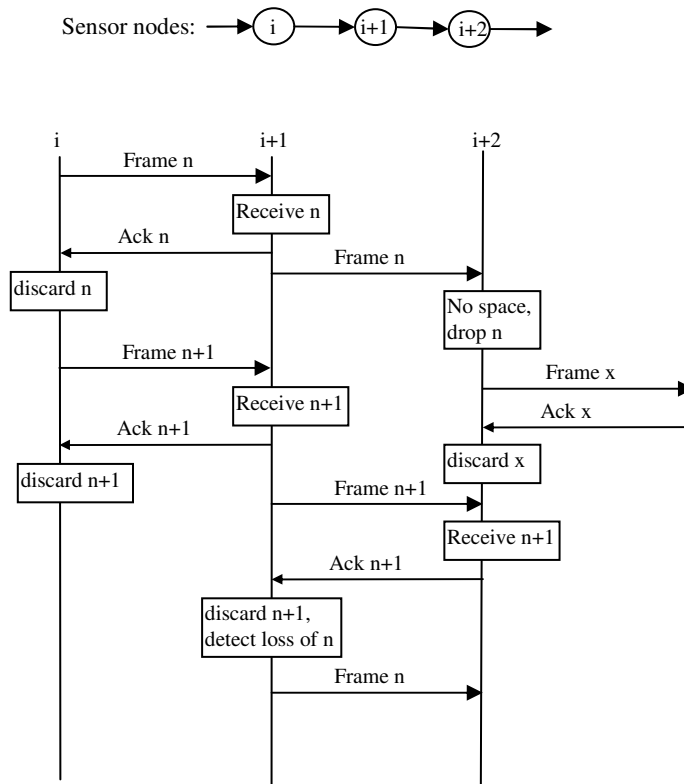


Figure 4-5. The end-to-end reliability scheme through hop-by-hop acknowledgement.

The one-source network structure shown in Figure 4-4(a) is used since we are mainly concerned about the video quality of hop-by-hop adaptation. The average bandwidth is 4.2Mbps, which is a little higher than the average bit-rate of the video stream, on all links except the bottleneck link. The bandwidth for the bottleneck link varies as shown along the x-axis in Figure 4-6. Also for the bottleneck link, there is a 6.7-second break starting at the 33rd second and a 16.67-second break at the 66th second.

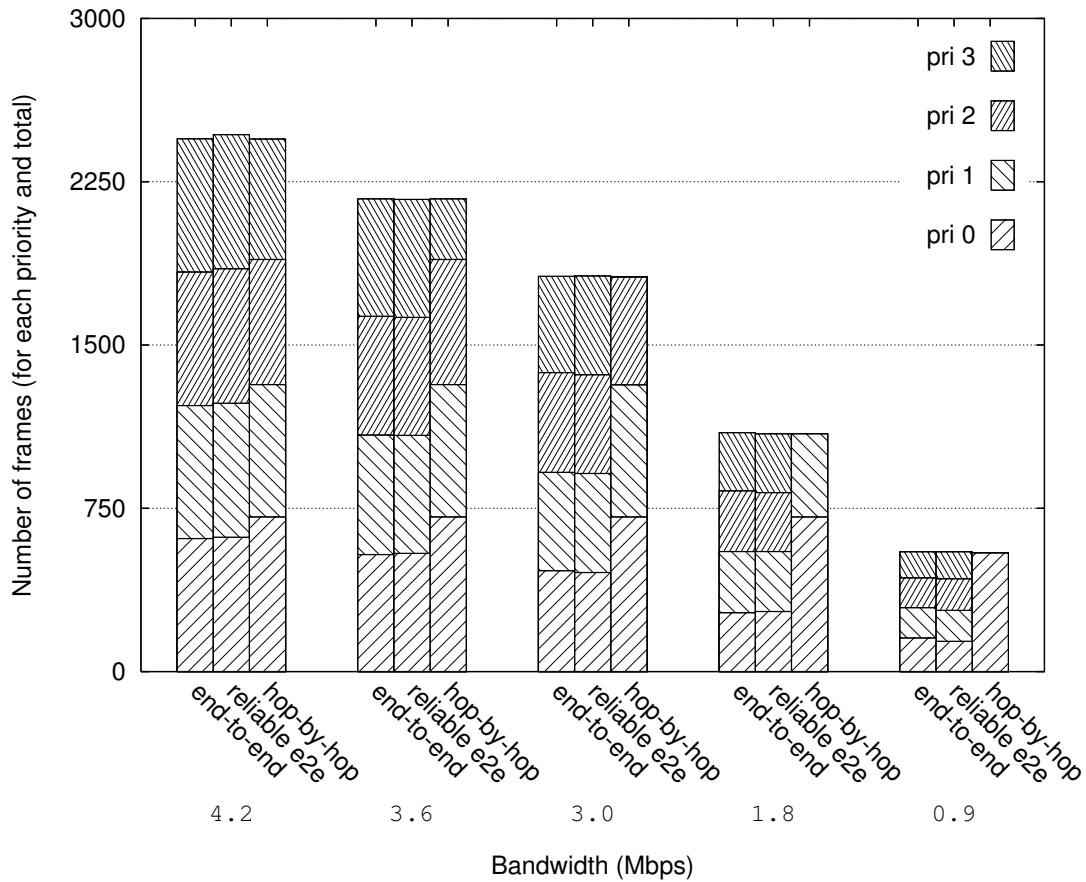


Figure 4-6 . Throughput and priority distribution for end-to-end adaptation and hop-by-hop adaptation. The height of a column represents the total number of frames received. There are at most four sub-parts within each column and each sub-part represents the number of frames for a certain priority level.

The priority distribution for end-to-end adaptation and hop-by-hop adaptation is shown in Figure 4-6. In Figure 4-6, the height of a column represents the total number of frames received. There are at most four sub-parts within each column and each sub-part represents the number of frames for a certain priority level, from the priority level zero at the bottom to the priority three on the top. Although the throughput is similar for both end-to-end adaptation and hop-by-hop adaptation, the hop-by-hop mechanism gets more high priority data through when the bandwidth is low. For

example, when the bottleneck link has bandwidth 1.8Mbps, ideally about 1,000 frames (one third of the total frames) can get through and 750 of which are of priority zero and 250 of which are of priority one. The throughputs for all three approaches are close to the ideal case. Only the hop-by-hop approach has a close to ideal priority distribution: about 710 priority-zero frames and 382 priority-one frames. The two end-to-end approaches have frames evenly distributed, about 250 frames for each priority level. If there are dependencies among these frames, the decodable frames for these two approaches are far more less than 1000 frames. The frame distribution along the time line is shown in Figure 4-7.

Figure 4-7 shows the frame rates for these approaches when the bottleneck bandwidth is 1.8Mbps. The frame rates are calculated based on the capturing timestamps, not on arrival time because there is no real-time requirement. Ideally, the frame rate should be about ten frames per second all the time during the simulation. None of the approaches achieve this frame rate. The hop-by-hop, however, mechanism has a much smoother frame rate than the two end-to-end adaptation mechanisms because it effectively utilizes buffer space on multiple nodes for adaptation. Without reliability, the end-to-end approach has random frame rates because the network drops data randomly; with reliability, only frames of the first 33 seconds get through because the video source makes adaptation decisions based on the

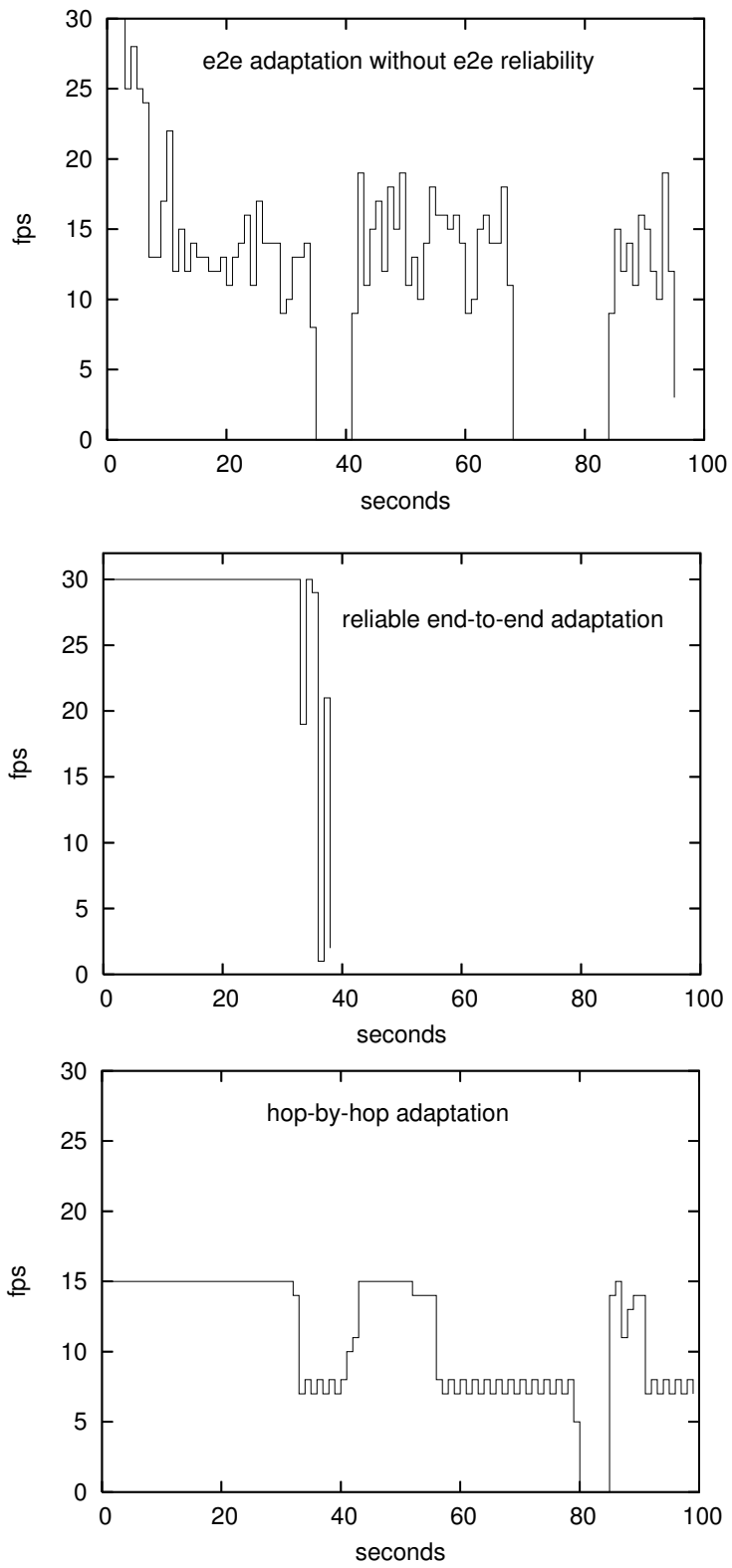


Figure 4-7 The frame rates. The frame rates are calculated based on the capturing timestamps, not on arrival time.

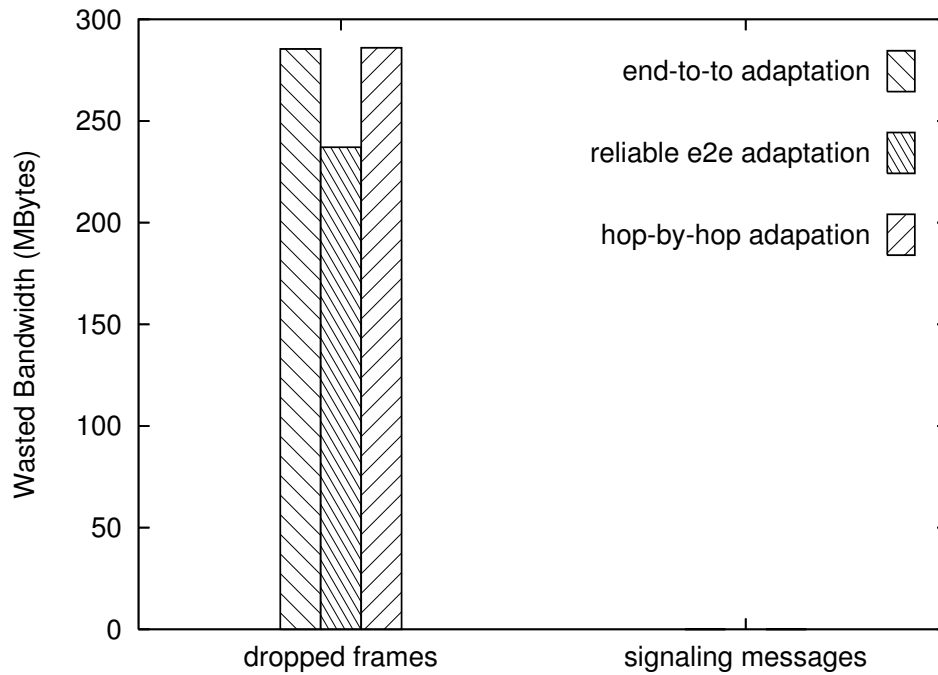


Figure 4-8 Wasted bandwidth

bandwidth on the first hop and tries to send out data with all priorities while the reliability scheme uses up the available bandwidth to send the first 33-second data. .

Figure 4-8 shows the wasted bandwidth for end-to-end adaptation and hop-by-hop adaptation. The signaling traffic for end-to-end reliability is negligible. The dropped data for end-to-end adaptation without reliability and hop-by-hop adaptation are similar because they both send data aggressively. End-to-end adaptation with reliability uses much of the buffer space for unacknowledged data thus slows down the sending and drops about 50 megabytes (17%) less data than the other two mechanisms. However, we do not believe that the saving can justify its highly variable frame rate.

Simulations in this subsection clearly show that hop-by-hop adaptation can achieve better video quality on a multi-hop route than end-to-end adaptation. If there are dependencies among frames such as in MPEG, the advantage of our approach is more significant because a dropped high priority frame can cause many low priority frames to be un-decodable. In the next subsection, we study different ways to compose mechanisms on individual nodes in the hop-by-hop framework to better video quality, to reduce bandwidth waste, and to promote fair sharing.

4.3.3 Exploring *Steens*

In this subsection, we experiment with different buffer management schemes and signaling protocols and study their effects on the composed adaptation system. We first study the video quality and bandwidth waste for a single source; then we study the bandwidth sharing among multiple sources.

4.3.3.1 Effective adaptation for a single source

In this subsection, we focus on the effect of signaling protocols on hop-by-hop adaptation for a single source assuming the single source can use all available buffer space. The network structure and the network bandwidth are the same as the simulations in Section 4.3.2, where the last link to the sink is the bottleneck link and has two breaks during the simulation. We assume signaling messages are reliably transmitted; however, the signaling protocols still work even if signaling messages are lost occasionally. For example, in case a “buffer full” message is lost, the destination node of this message keeps sending data to the full node, which will be dropped at the full node and trigger the sending of another “buffer full” message.

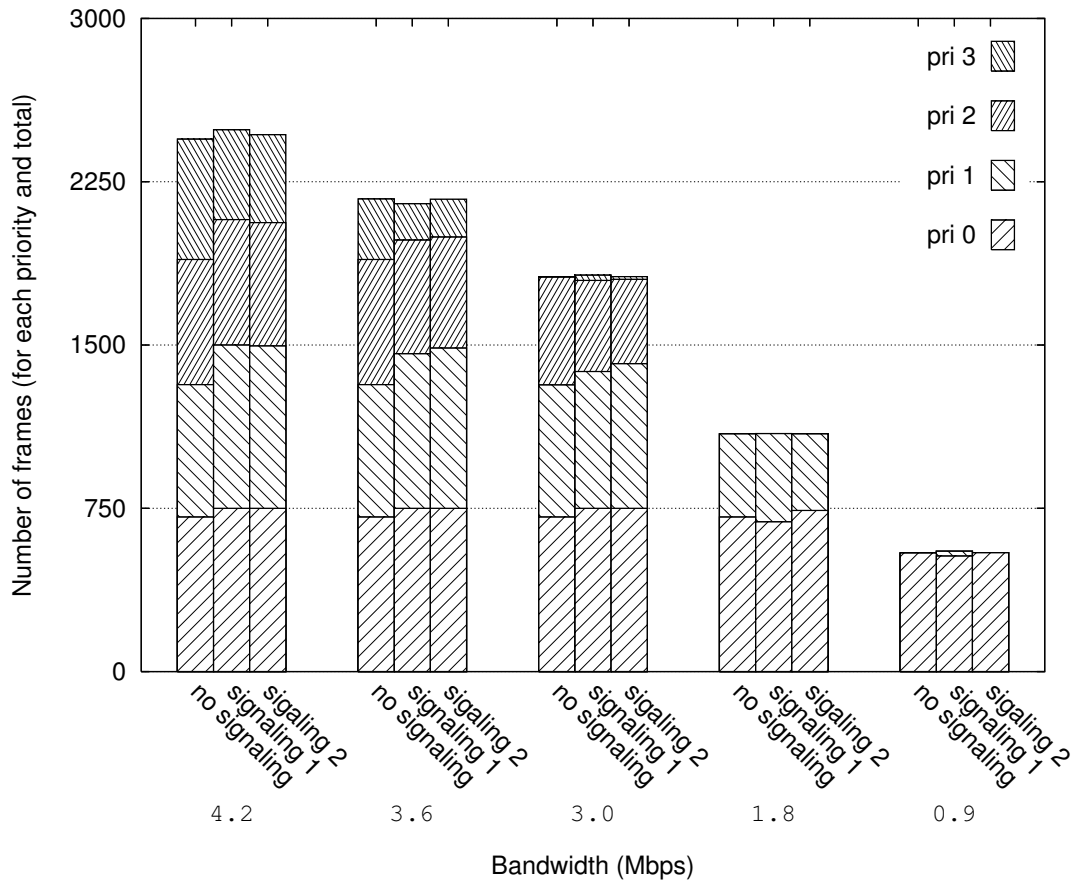


Figure 4-9 . Throughput and priority distribution for three hop-by-hop adaptation systems. The height of a column represents the total number of frames received. There are at most four sub-parts within each column and each sub-part represents the number of frames for a certain priority level.

The throughput and the priority distribution are shown in Figure 4-9 for three hop-by-hop adaptation systems: without explicit signaling, with *Signaling 1* that exchanges “buffer full” messages, and with *Signaling 2* that exchanges dropping levels. The throughput is almost the same for all three systems, as expected. In most cases, the two systems with signaling have more high priority data than the system without signaling because they delay the dropping of high priority data in a full buffer by pushing dropping towards the source. The number of priority inversions is reduced because priority inversions occur when there are multiple nodes dropping frames and

having different dropping levels. However, for *Signaling 1*, at the initial stage of the simulation, there are low priority data getting into intermediate nodes and they cannot be replaced by high priority data when the buffer is full. This low priority data gets sent even when the average bandwidth is very low. In general, this is not a problem for *Signaling 2* because it allows high priority data to enter a full buffer to replace those low priority frames unless occasionally these low priority data get sent before high priority data arrive.

Figure 4-10 shows the frame rates for the bottleneck bandwidth of 1.8Mbps. Still, no approach achieves the ideal 10 frames per second. However, both systems with signaling do better than the system without signaling. *Signaling 1* smoothes the frame rate over the first short break; with *Signaling 2*, the frame is smoothed into two relatively stable phases: before the 45th second, the frame rate is 10 frames per second; after that the frame rate is about 7.5 frames per second despite the long break on the bottleneck link. The cause of the difference lies in how the buffer space is used as the smoothness of video is tightly coupled with the buffer space for adaptation. In all three cases, buffer space on intermediate nodes is used for adaptation. However, without signaling, dropping starts immediately when the bottleneck link breaks while the two signaling protocols delay the dropping by asking other nodes in the network to store some data so the buffer space is used more efficiently than without signaling.

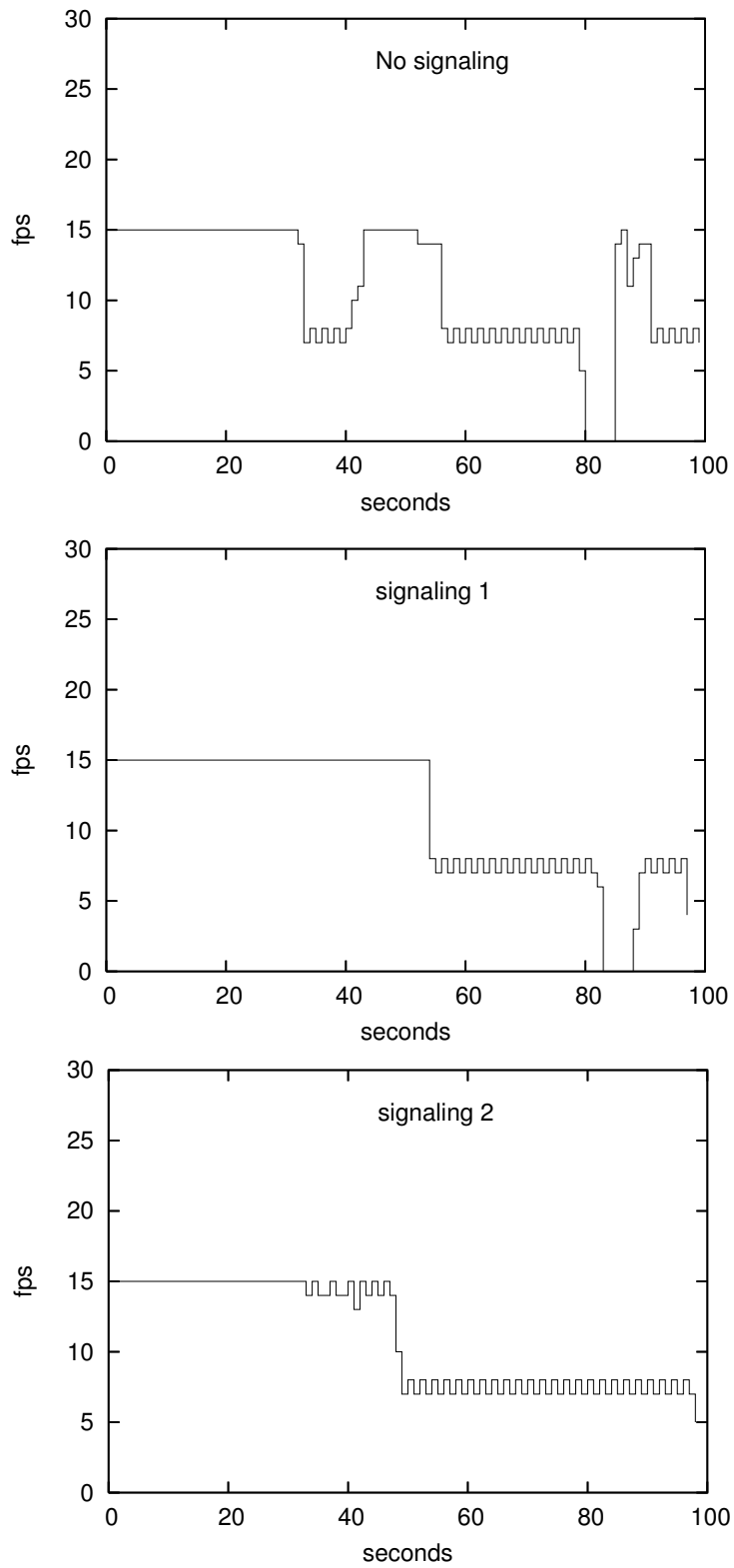


Figure 4-10 The frame rates (hop-by-hop adaptation). The frame rates are calculated based on the capturing timestamps, not on arrival time.

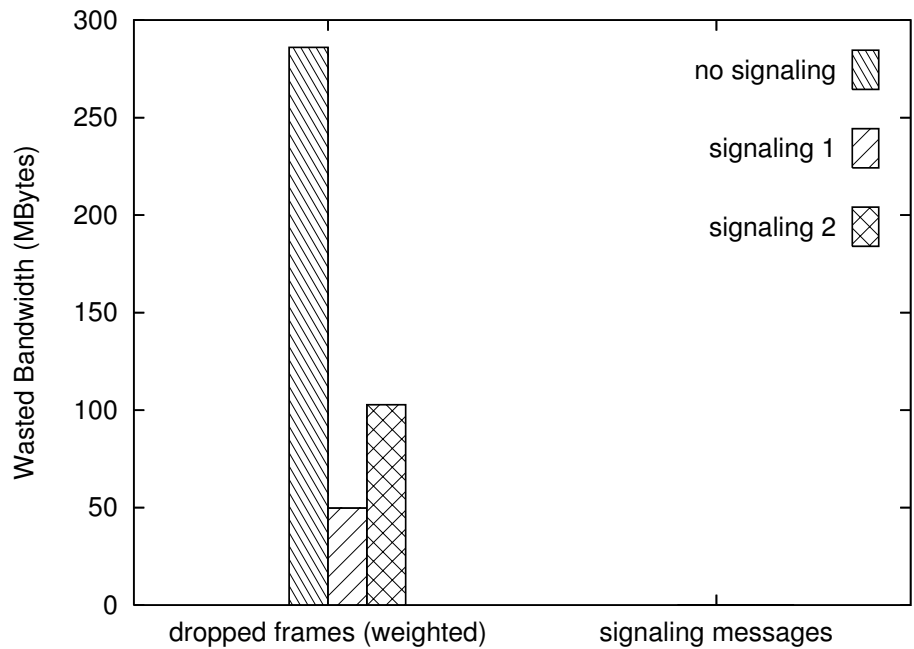


Figure 4-11 Wasted bandwidth

Signaling 2 uses the buffer space more efficiently than *Signaling 1* because it not only uses the buffer space to store data but uses it to store high-priority data by trying to maintain a system-wide dropping level.

Next, we show the reduced bandwidth wastage through signaling (the bottleneck bandwidth is 1.8Mbps) in Figure 4-11. *Signaling 1* and *Signaling 2* greatly reduce the amount of data dropped in the network, 82.7% and 67.1%, respectively. The price they pay is negligible: 7620 and 8850 signaling messages. Assuming 20 bytes per signaling message, the wasted bandwidth for signaling messages is just about one video frame, 0.053% and 0.062% of without-signaling-bandwidth-wastage.

For *Signaling 1* and *Signaling 2*, we need to decide the high watermark and the low watermark for the congested node. In Figure 4-9 and Figure 4-10, the high watermark is that the buffer is completely full and the low watermark is that the buffer has 10% free space. That is, in *Signaling 1*, a node asks its upstream nodes to stop sending when the buffer is full and to resume sending when the buffer has 10% free space; in *Signaling 2*, a node will decrease the dropping level when the buffer is full and increase the dropping level when the buffer has 10% percent free space. Obviously, changing the watermarks may change the behavior of the adaptation. We have experimented with the low watermark of 5%, 20%, 30%, and 50%. The priority distribution, the frame rate, and the wasted bandwidth are shown in Figure 4-12 to Figure 4-17 for the two signaling protocols respectively. Changing the watermarks has a significant impact on *Signaling 1*. In Figure 4-12, it is obvious that more low priority data are protected and sent by a larger low watermark because it delays the arrival of high priority data from other nodes. For example, when the bottleneck bandwidth is 1.8Mbps, the low watermark 50% has about 164 less priority-zero frames but 183 more priority-two frames than the low watermark 5%. Figure 4-13 shows clearly that more early data are protected by a larger low watermark regardless of their priority. When the low watermark is 50%, the frame rate for the first 20 seconds is about 22.5 frames per second but falls to zero after about the 73rd second. A large low watermark does reduce wasted bandwidth as shown in Figure 4-14. For example, the low watermark 50% wastes 70% less bandwidth than the 5% watermark. This is expected since a large low watermark causes less aggressive sending and less

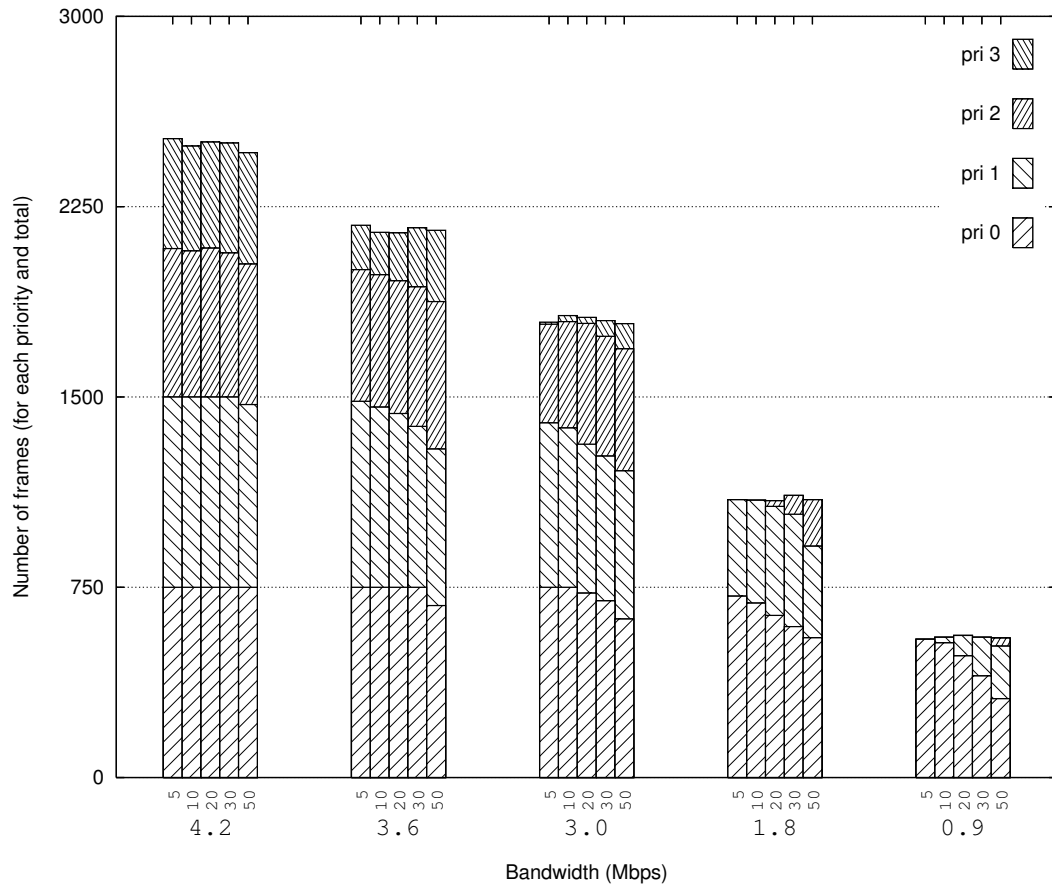


Figure 4-12 . Throughput and priority distribution for *signaling 1* with different low watermarks. The height of a column represents the total number of frames received. There are at most four sub-parts within each column and each sub-part represents the number of frames for a certain priority level.

oscillation thus less data dropping than a small low watermark. Increasing the low watermark for *Signaling 2* also increases the number of low priority data in the sink and reduces the wasted bandwidth as shown in Figure 4-15, Figure 4-16, and Figure 4-17; however, it is not as sensitive to the low watermark as *Signaling 1* because it always allows high priority data to get through.

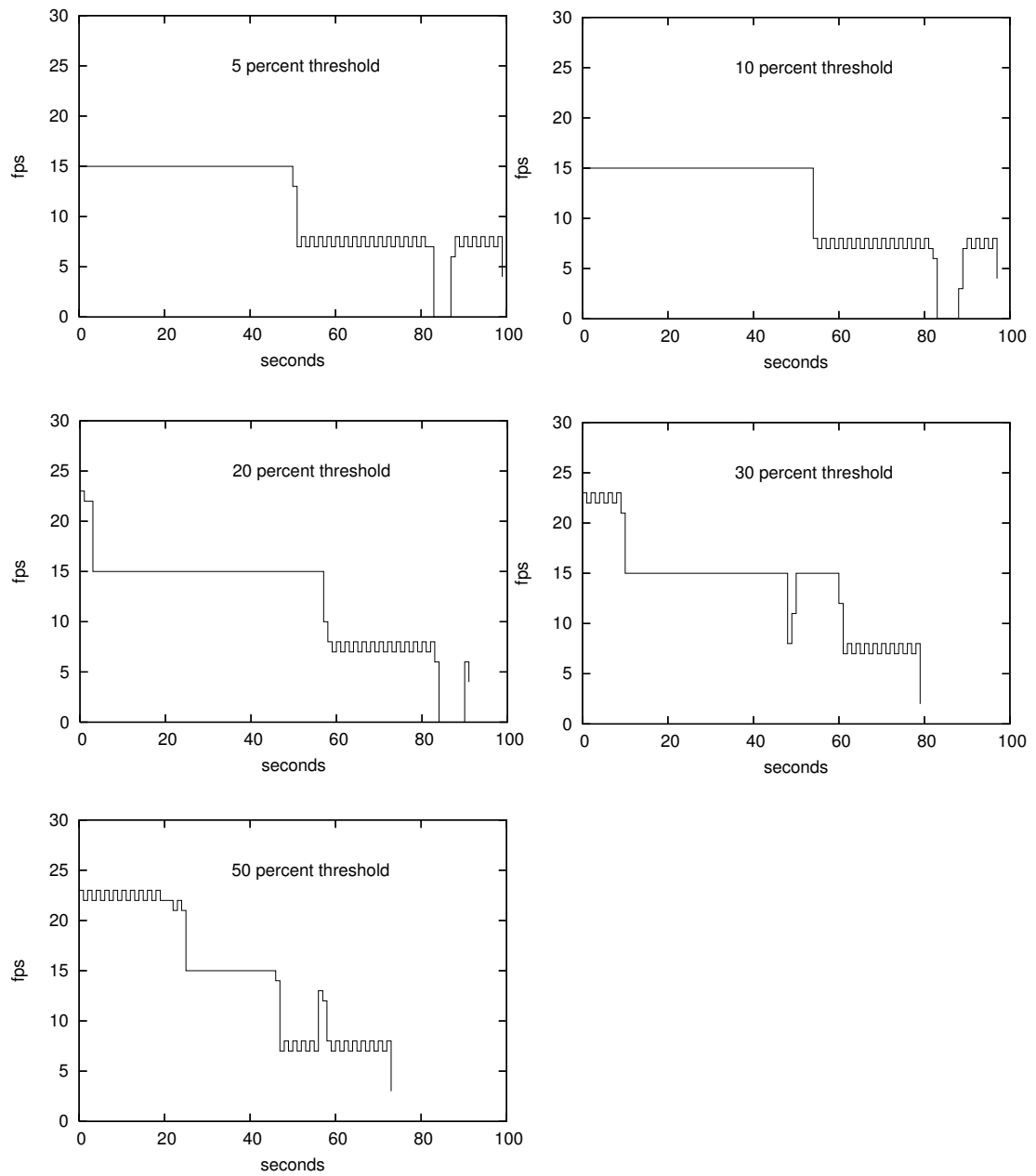


Figure 4-13 The frame rates (*signaling I*). The frame rates are calculated based on the capturing timestamps, not on arrival time.

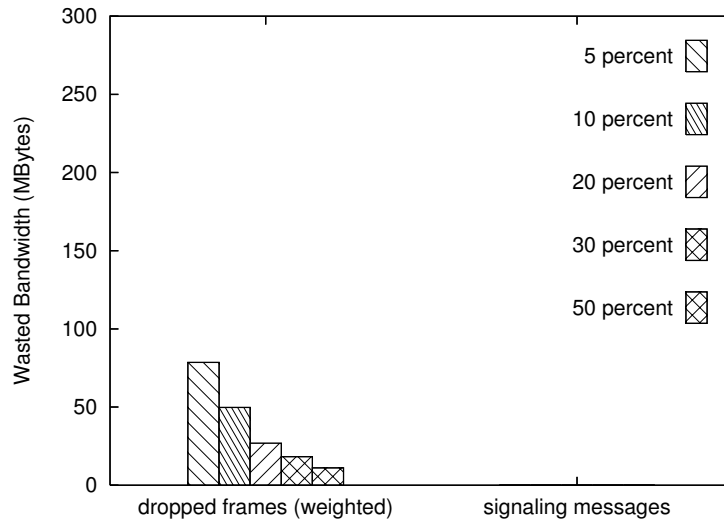


Figure 4-14 Wasted bandwidth

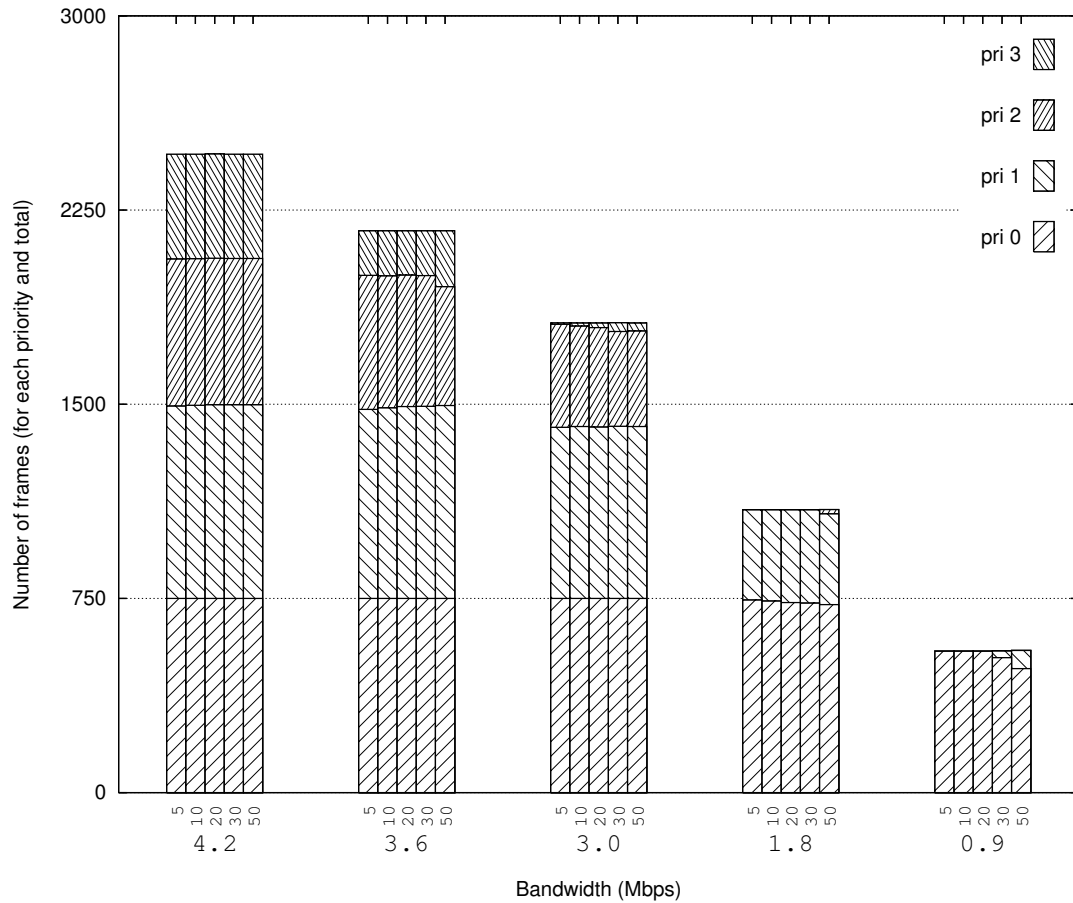


Figure 4-15 . Throughput and priority distribution for *signaling 2* with different low watermarks. The height of a column represents the total number of frames received. There are at most four sub-parts within each column and each sub-part represents the number of frames for a certain priority level.

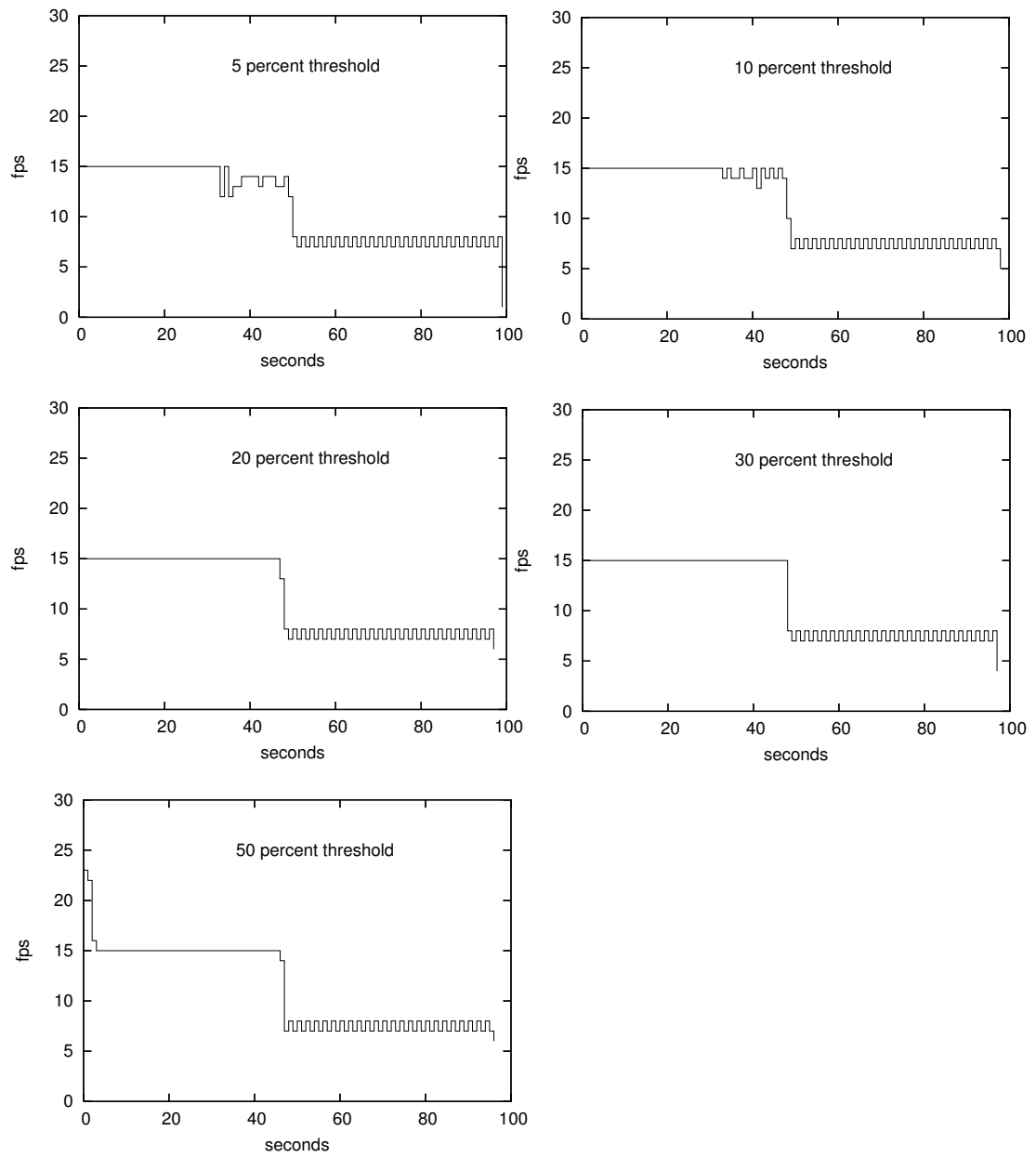


Figure 4-16 The frame rates (*signaling 2*). The frame rates are calculated based on the capturing timestamps, not on arrival time.

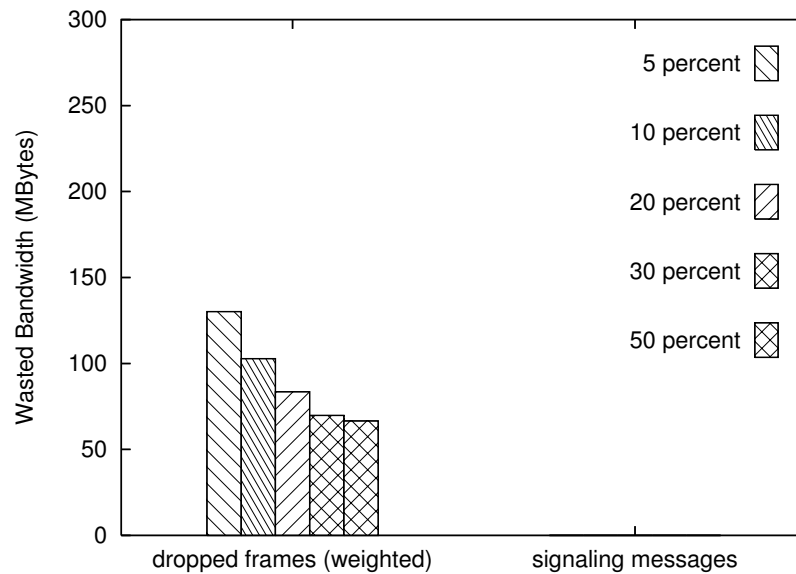


Figure 4-17 Wasted bandwidth

Changing the watermarks provides a means to make trade-offs between video quality and wasted bandwidth. For example, for *Signaling 2*, the low watermark 10% and the low watermark 30% result in similar video quality while the low watermark 10% leads to about 30 megabytes more wasted bandwidth.

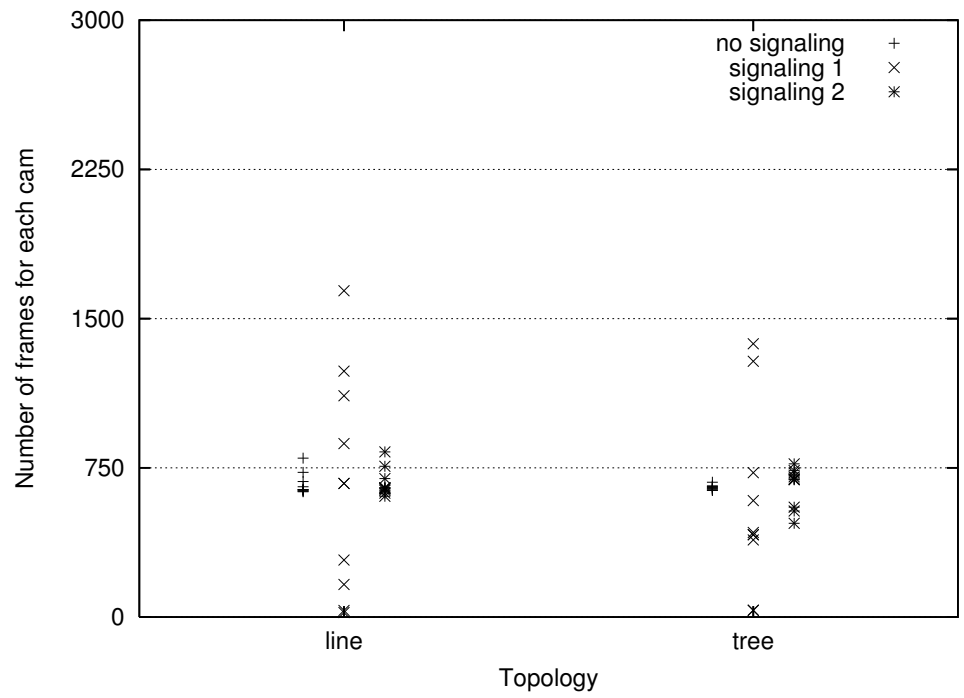
4.3.3.2 Bandwidth sharing among multiple sources

The focus of this subsection is on bandwidth sharing among multiple sources. Intuitively, the network topology may change the sharing because sensor nodes farther away from to the sink are likely getting less bandwidth on the bottleneck links, which are usually close to the sink. Therefore, we use two network structures, both having ten cameras, and try to draw conclusions independent of network topology: one is the line structure shown in Figure 4-4(b) and the other is the tree structure in Figure 4-4 (c)

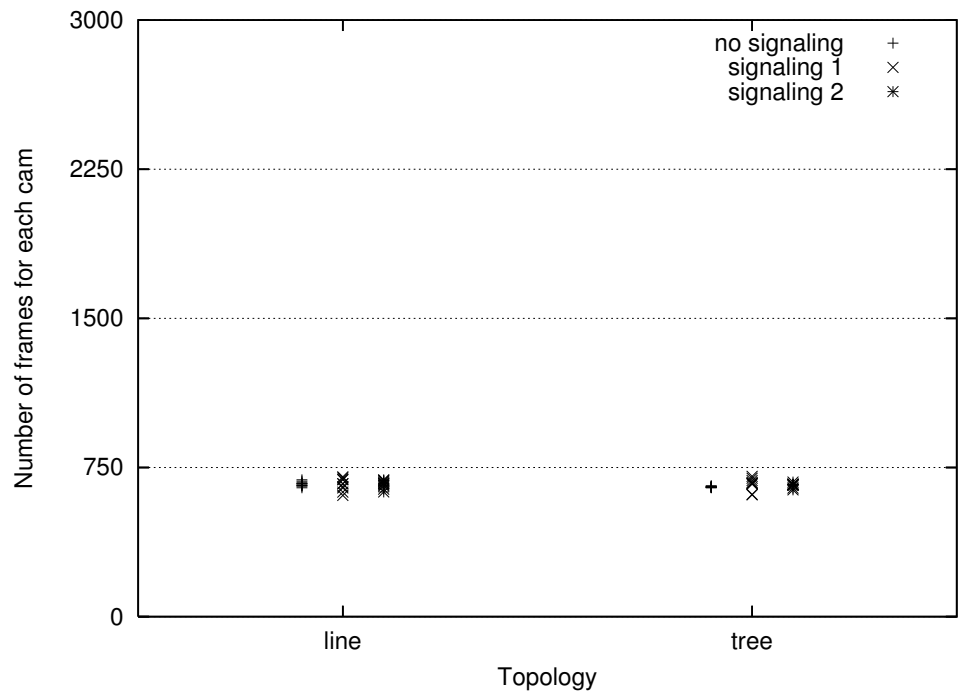
The bottleneck link in both structures is the last link to the sink. All links have 13.86Mbps bandwidth, which allows about frames from three cameras to get through. The bottleneck links have four breaks at the 16th second, the 23rd second, the 50th second, and the 66th second with a total break time of 33.33 seconds, which are one third of the simulation time.

In this chapter, we assume that all sources have the same priority, the same prioritization mechanism, and the same data rate. The total bytes can be transmitted on the bottleneck link is about 115 megabytes, that is, about 6,683 frames. Therefore, each source should have about 668 frames in the sink under equal sharing, preferably weighted toward higher priorities.

Figure 4-18 shows the numbers of received frames for each camera, for first-come-first-serve buffering and partitioned buffering, respectively. Partitioned buffering enforces fair sharing, regardless of the topology or the signaling protocol, as shown in Figure 4-18(b). The maximum standard deviation with partitioned buffering is 36.32 when *Signaling 1* is used in the line structure. For the same signaling protocol and topology, the standard deviation is 522.56 without partitions because *Signaling 1* is very biased to nodes close to the sink whose data arrive early at *node 9*. Without signaling, the standard deviation is only 52.73 because all nodes send aggressively and there is enough bandwidth to transmit their data to *node 9* to compete for the bandwidth on bottleneck link. *Signaling 2* also protects early data but it allows high priority data to be sent to a full buffer and forcing low priority data in the full



(a) no partitions



(b) with partitions

Figure 4-18. Fairness. Received frames for each camera.

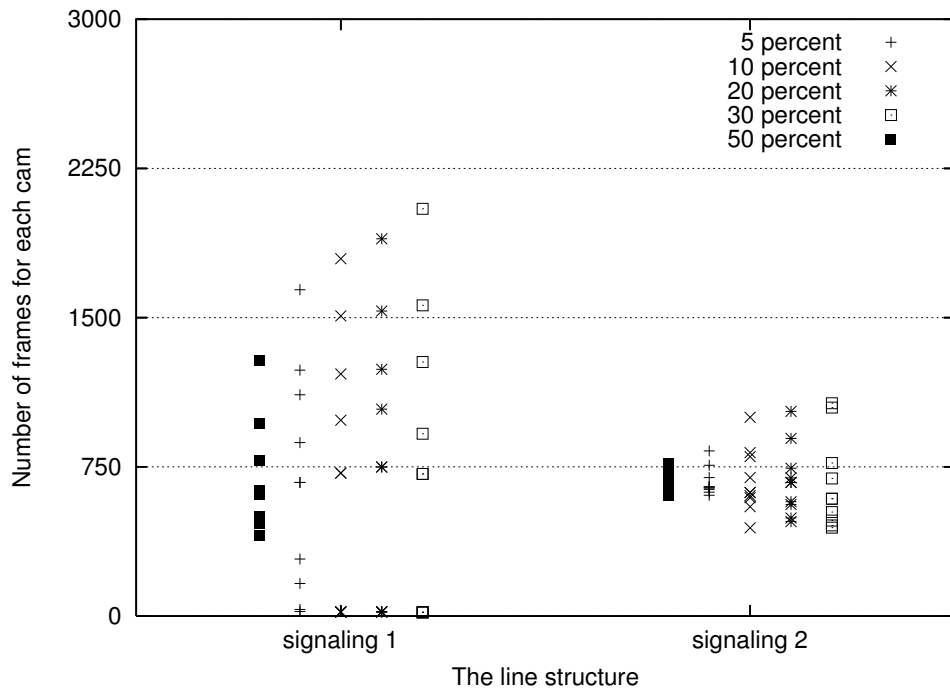


Figure 4-19. Fairness. Received frames for each camera.

buffer to be dropped; therefore the sharing is greatly improved compared to *Signaling 1* and the standard deviation is 71.11. The tree structure is a little more amenable for fair sharing because the difference in number of hops is smaller in the tree structure.

Changing the low watermark for resuming receiving in *Signaling 1* and the low watermark for decreasing the dropping level in *Signaling 2* can change the sharing among cameras when the buffer is not partitioned. In Figure 4-19, we show the effect of changing the low watermarks on fair-sharing in the line structure. In general, a smaller low watermark means more aggressive sending thus better sharing. The standard deviation for *Signaling 1* ranges from 257.7 to 723.94 as the low watermark goes from 5% to 50% and the standard deviation for *Signaling 2* ranges from 43.33 to 218.82. Nevertheless, *Signaling 1* is bad for sharing whatever the low watermark is.

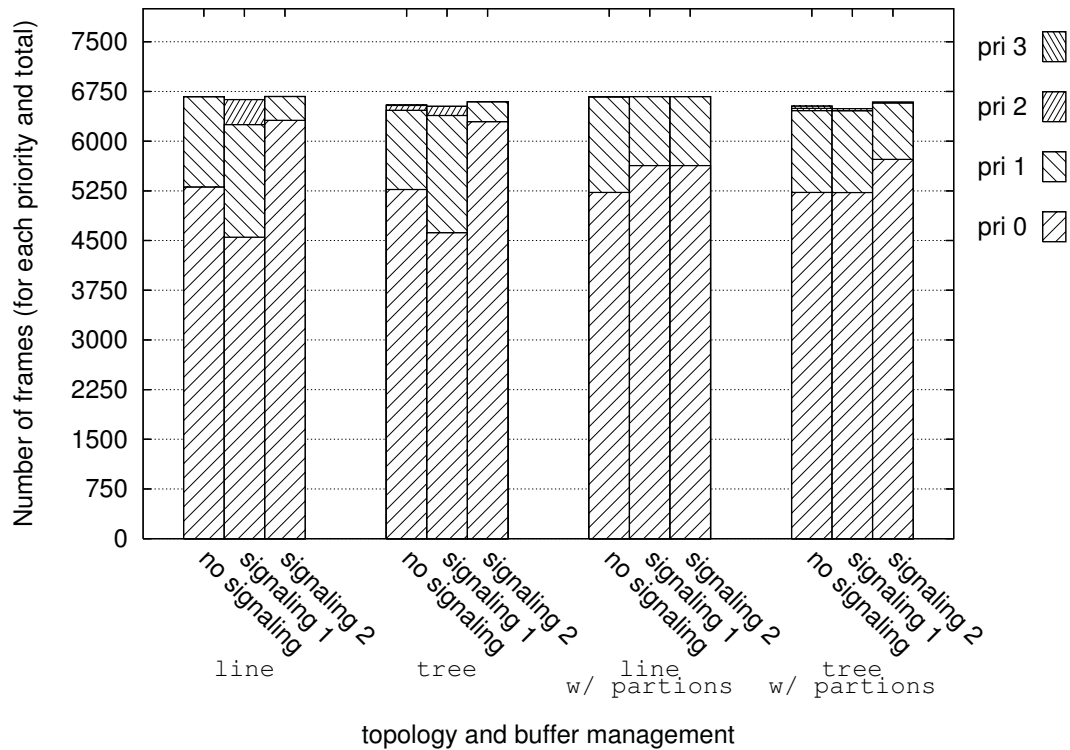


Figure 4-20. Throughput and priority distribution. The height of a column represents the total number of frames received. There are at most four sub-parts within each column and each sub-part represents the number of frames for a certain priority level.

Figure 4-20 shows the priority distribution for multiple video sources. In Figure 4-20, partitions do not lower the throughput in our configuration because no partition is underflowing. *Signaling 1* has better priority distribution with partitions because low priority data are forced to be dropped even at the beginning stage due to the small partition and less low priority data can get to the sink. *Signaling 2* works better without partitions. For example, the tree topology without partitions has 6294 priority-zero frames, which is 10% more than with partitions. We believe that this is because high-priority data cannot get into a buffer when its partition is full while other partitions have lower priority data.

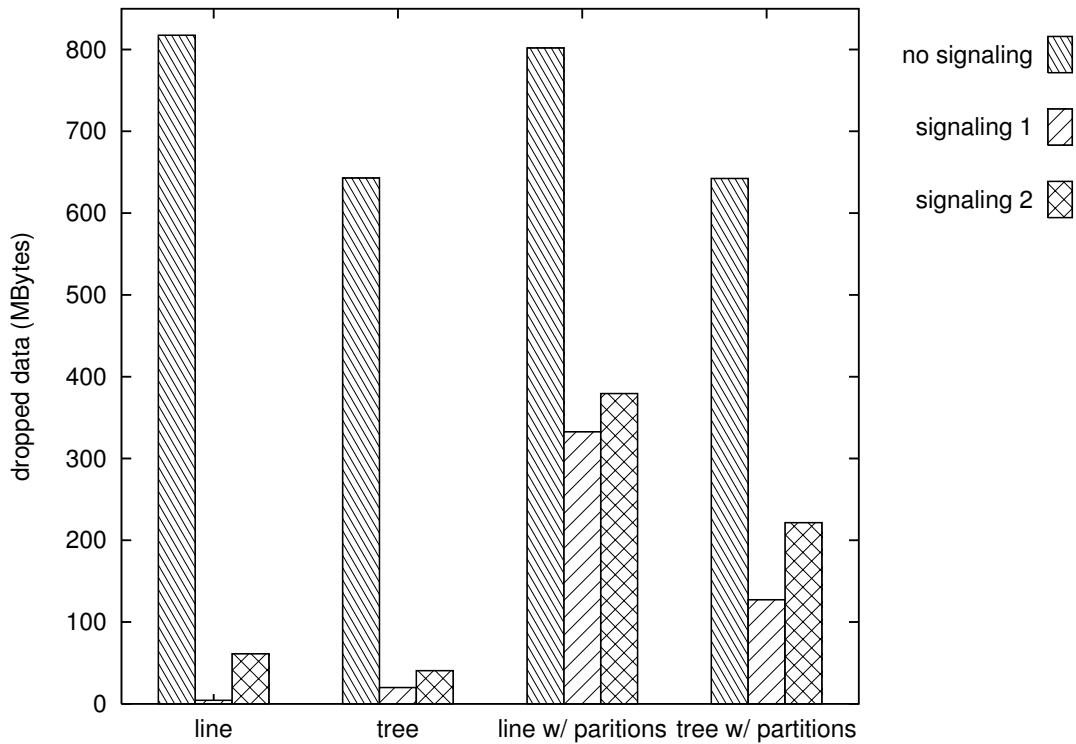


Figure 4-21 Wasted Bandwidth (dropped data)

Figure 4-21 shows the wasted bandwidth for multiple resources. As the space for each partition is small, thrashing occurs for *Signaling 1* and *Signaling 2*, which means more data dropping as shown in Figure 4-21. For example, for the line structure and *Signaling 2*, partitioning drops eight times more data than first-come-first-serve buffering.

It is worth noting that in our experiments, buffer partitions perfectly reflect network topology. In a real sensor network, video sources may have different data rates and the network topology is always changing. A system with partitioned buffer may not be able to achieve the fair-sharing shown in this section and buffer underflow

can happen, which lowers the throughput of the system. Therefore, we believe that *Signaling 2* without partitions is a good option for fair sharing.

4.4 Conclusions

In this chapter, we propose *Steens*, a multi-hop buffering and adaptation framework for video-based sensor networking applications. We have shown that adapting video in the network is more effective in collecting high quality video than adapting video at the network edges. We also show that properly sharing information among sensor nodes can achieve smoother frame rates and reduce bandwidth wastage. Sharing of application-information information among nodes can also maintain fair sharing of bandwidth.

CHAPTER 5

CASCADES: SUPPORTING VIDEO ADAPTATION IN SENSOR NETWORKS

Providing video adaptation within a sensor network requires the underlying systems software to support some degree of programmability and retaskability while retaining high performance. In this chapter, we describe *Cascades*, a flexible component-based framework for multi-modal sensor networking applications. We also describe how it supports video adaptation within a sensor network.

5.1 Introduction

In order to reduce the power consumed for communication and to maximize scalability of a sensor network, it is necessary to process or filter the data from various sensors as close to the source as possible. For some applications, this might be at the sensor itself, while in other applications it might be at a point where several sensor data streams are fused together. Programming a distributed, embedded, and heterogeneous sensor network system consisting of up to thousands of sensor nodes is a formidable challenge. This is further complicated by the fact that the processing within the network may need to be adjusted or changed because the sensor application may be dynamic over time. Changes might be in response to an event captured within the sensor network or new algorithms being developed by the user to assimilate data.

Therefore, supporting easy-to-program and easy-to-retain data processing in the network is essential to the cost-effective development and maintenance of sensor networking applications.

In scalar sensor networks, programmability and retaskability are most often provided by query language interfaces that provide high-level abstractions for programming. TinyDB [51][52], Cougar [86], and SINA [68] are representative examples of the query language approach. They view the sensor network as a distributed database system and queries are distributed and processed in the network automatically. Programming and retasking are accomplished by simply sending queries to the sensor networks. However, the query language approach is suitable only for *scalar* sensor networks because scalar data can be adequately processed through generic operations such as MAXIMUM, MINIMUM, and AVERAGE.

We propose *Cascades* [36], a flexible component-based framework, to provide programmability and retaskability to process multimedia data in the network. Unlike scalar data, multimedia data processing tends to be very application-specific and cannot be implemented solely through generic operations. Instead, component-based approaches are often used to facilitate the development of multimedia applications by flexibly reusing complex algorithms. For example, the Continuous Media Toolkit (CMT) [56] from Berkeley allows users to construct streaming applications rapidly through TCL scripts that combine lower-level video-based components. *Cascades* adopts the component-based approach and addresses sensor networking issues such as

retaskability and the integration of scalar data and multimedia data. We believe that such a framework is necessary for building multimedia sensor networking applications.

In this chapter, we examine whether video adaptation in sensor networks can be accomplished efficiently in a component-based framework. One key focus of our work is the need for high-performance. Given the power and processing constraints on embedded devices, component frameworks that are slow may not be usable in a sensor network setting. We will show how *Cascades* can meet the requirements of programmability, restaskability, and high performance. We will first describe the overall architecture of *Cascades* in Section 5.2.

5.2 The *Cascades* Architecture

As with any other component-framework, *Cascades* needs to provide the ability to combine the components in a meaningful way. At one extreme, composability can be accomplished through pre-defined code segments that are compiled together into a single monolithic executable, allowing the system to run as efficiently as possible. Unfortunately, this eliminates the ability to make changes to a running system. At the other end of the spectrum, one could imagine using a shell-level scripting program to compose such a system from a number of smaller executables each running as a separate process. While making it easier to distribute smaller sub-components, such a system may suffer from a large amount of overhead in switching between address spaces and marshalling of data between stand-alone executables.

Cascades adopts an approach somewhere between these two extremes: it uses a high-level scripting language to connect highly-optimized components so that they execute in the same process. High-level scripting languages allow users to specify rather complex systems with minimal code. Furthermore, they allow programs written in high-level languages such as C or C++ to be called as part of the script. This allows a majority of computationally intensive code (such as video processing algorithms) to be written in a language with a highly optimized implementation.

In *Cascades*, we have chosen to use Python as the high-level scripting. We have several requirements for the scripting language. First, we prefer a language that supports Object-Oriented Programming because objects fit into a component-based framework naturally. Second, it needs to provide the complex data structures that are needed to manage multimedia data. Third, to support re-tasking, it must have the ability to add to or change the behavior of parts of the system while it is running. Finally, the combination of components needs to be of high-performance in order to minimize impact on systems performance. We have chosen Python because it meets all these requirements and it is available on the two embedded platforms in our test bed. We would expect that other scripting languages that meet these requirements to work as well.

The primary mechanism used to support the processing of multimedia data and the integration of multi-modal data in *Cascades* is cascading filters. Filters are user-supplied or toolkit-derived components that allow the sensor sub-system to tailor its

data for the user application. The idea behind filters is that they process data with a highly optimized piece of code (rather than with an interpreted language). There are several basic types of filters that we envision.

Efilters (error filters) are the primary mechanism by which the handling of faulty sensors can be specified. As an example, faulty readings can occur from bio-fouling of the sensors in outdoor scenarios. These filters can consist of standard statistical filtering techniques; they can also allow the application to specify the exact way in which the faulty data may be handled.

Dfilters (scalar data filters) are used to manage scalar data within the sensor network. They take one or more streams of scalar data as input and produce as output one or more data streams as well as meta-information about the data. As an example, one filter might calculate the average value measured per hour, either for a single sensor or a group of sensors. The filter might also add meta-information such as timing information or relational information between sensors. The sensor output can then be used by other filters.

Vfilters (video filters) are used to manage video data being collected by video sensors. Vfilters might consist of application-specific video processing algorithms or off-the-shelf components. Application-specific algorithms might include image-processing techniques for object identification; an off-the-shelf component might include a compression algorithm or video adaptation algorithm.

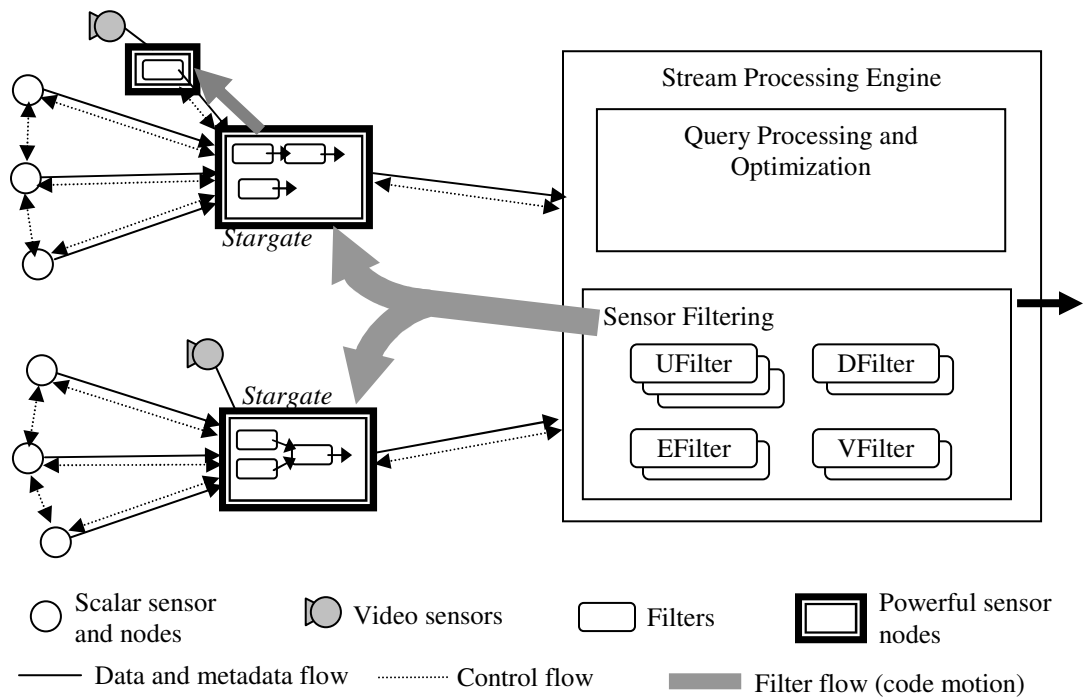


Figure 5-1 An example system in the framework of *Cascades*. The *Stargate* nodes are more powerful than scalar sensor nodes and can both capture video and be used to manage a number of scalar sensor nodes.

Ufilters (*user filters*) are user-specified filters that allow the user to specify the integration of data from the other types of filters, for example, the annotation of video streams using scalar sensor data.

The focus in this thesis is on the video filter aspects of the component-based framework. We leave other types of filters as future work.

An example system in the *Cascades* framework is shown in Figure 5-1. We believe that large multi-modal sensor networks will have a multi-tiered architecture that consists of low-power sensor nodes such as the scalar sensor nodes in Figure 5-1 and high-power sensor nodes such as the video sensor nodes. The scalar sensors nodes,

which are not capable of running Python due to memory constraints, can run the operating system of their choice such as TinyOS [33]. The scalar sensors can be abstracted to the point that they can be plugged as input into a *dfilter* into the *Cascades* systems. For example, *Cascades* provides a *mote abstraction layer*, which encapsulates the functionality of TinyDB [51] and exports the data collected from Berkeley motes [33] through generic Python interconnects. The mote abstraction layer runs on sensor nodes that are capable of data aggregation, so data from Berkeley motes can be collected by other filters without dealing with the communication details or TinyDB interfaces. *Crossbow Stargates* in Figure 5-1 are an example platform for such data aggregation tasks. These nodes are powerful enough to process video data and to support *Cascades*' cascading filter architecture. In the *Cascades* framework, the base station of a sensor network contains a stream processing engine that determines the filters needed and their locations. A filter management system will transmit and load filters into sensor nodes dynamically.

We do not expect *Cascades* to be a complete system. Rather, we are interested in providing a framework for others to use in order to gain insights into what abstractions are needed for building multimedia sensor networking applications. We believe that once a large number of example applications have been assembled, it will be much easier to provide a polished, generic, and relatively complete middleware system.

5.3 Implementing Video Adaptation in Cascades

In this subsection, we describe our implementation of a simple prototype of *Steens* in the *Cascades* framework. The prototype includes video capturing, video filtering, video compression, video adaptation, and collaborative signaling. The structure of this system is shown in Figure 5-2. The two video sources are both Panoptes video sensors [19], which use the *Crossbow Stargate* embedded sensor platforms. The *Stargate* platform runs the embedded Linux operating system 2.4.19-rmk7-pxa2. It has a 400 MHz Intel Xscale processor, 64 megabytes of memory, a 100 Mbps Ethernet connector, and a compact flash wireless 802.11 card. Video capture is accomplished through a Logitech *QuickCam* 4000 Pro USB camera. The in-network manager and forwarder is an Intel *StarEast* node that has a 533MHz Intel IXP425 network processor and 256 megabytes on-board SDRAM (only 64 megabytes are used). It runs Snapgear Linux 3.1.1, a *uClinux* distribution for embedded systems. Wireless communication is through an Intel Calexico II card. Although the *Stargate* nodes and the *StarEast* node both run embedded Linux, they are heterogeneous and two different cross compilers are needed. The sink is a Compaq laptop computer running Redhat 9.0. We will show in the following subsections how the software is constructed, how retasking is realized, and the performance overhead of the *Cascades* framework.

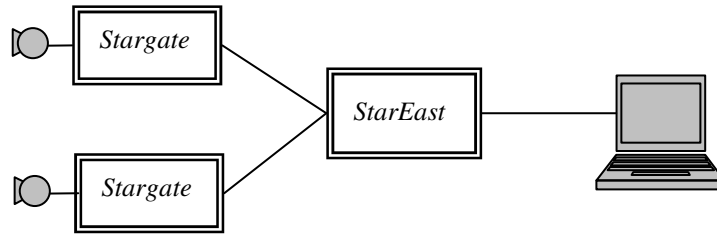


Figure 5-2 A simple adaptive video collection system. The two video sources are *Stargate*-based. The *StarEast* node is an in-network manager and forwarder. The laptop is the sink.

5.3.1 Programmability

We begin with a basic adaptive video collection system without collaboration among nodes. Our system consists of code from the original Panoptes video sensor[19], the *ffmpeg* MPEG-1 codec [22], and code that we wrote to bring them together. We implemented the prioritization mechanism in described in Figure 4-1. A majority of the code was in written in C or C++. We wrapped the C/C++ code segments with Python interfaces so that we have filters for capturing, motion detection, compression, prioritization, and networking. With these filters, we can quickly build the software on the video source nodes as shown in Figure 5-3. An example Python script connecting filters together is shown in Figure 5-4. In Figure 5-4, the application-layer filter we use is a motion filter, that is, video without change is discarded, which is very useful to reduce data processing and transmission for applications such as video surveillance. Other filters such as content-based filters can be plugged in as easily, assuming they have been written. The compression filter is a JPEG encoder and can be replaced with an MPEG encoder on-the-fly. The secretary filter packs and unpacks network messages and the messenger filter sends and receives

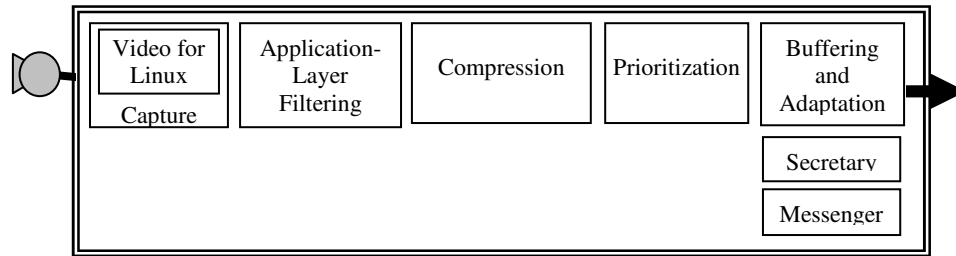


Figure 5-3 The construction of a video sensor capturing and adaptation system. Each of the filters has a Python interface, allowing it to be arranged in a variety of ways. The secretary filter packs and unpacks network messages and the messenger filter sends and receives messages. The Python script for an example video capture system is shown in **Figure 5-4**.

```

#Initialization and minor procedures removed

while 1:
    messenger.PollSockets()
    rawImage = camera.CaptureOneFrame()
    if motionDector.HasMotion(rawImage):
        JPEGImage, len = compressor.Compress(rawImage)
        priority = prioritizer.PrioritizeCircle();
        JPEGMsg = secretary.MakeJPEGMsg(JPEGImage, len, priority)
        buffer.PutMsg(JPEGMsg)
        msgToSend = buffer.GetNextMsgToSend()
        if messenger.SendMsg("manager", msgToSend, -1):
            buffer.RemoveSelectedMsg()
  
```

Figure 5-4 The capturing and adaptation script. In this example, the application-layer filter is a motion filter; the compression filter is a JPEG encoder; the prioritization mechanism is show in Figure 4-1; and the buffer sends messages in the priority order.

messages. The software on the *StarEast* node is composed in a similar way with fewer filters (no capture, motion detection, or compression filters). The prioritizer on the *StarEast* node does global prioritization that maps local priorities to global priorities.

We have also implemented collaboration and signaling protocols as the next step. Since Python itself can do complicated processing and access data structures defined in C++, we are able to implement signaling in Python and keep most of the C++ code unchanged. Figure 5-5 shows a sample Python script on the *StarEast* node for the ECN-like signaling in *Steens*. In the Python script, the manager node checks its buffer-fill level periodically. If the buffer status changes, it sends the “stop sending” message or the “resume sending” message to the video source nodes. Because Python is interpreted and does not need compilation, adaptation parameters such as the threshold for resuming sending can be adjusted easily. We have found that compiling for heterogeneous platforms is tedious and error-prone even though we have only two different platforms. The use of Python reduces the time required for prototyping, which requires frequent code changes. After the signaling implementation in Python was working properly, we ported it into C++ and built new buffering and adaptation filters that were capable of sending and receiving signaling messages and collaborating with other nodes.

In summary, we have found that the *Cascades* framework is quite useful for quick construction of our video adaptation prototype through reusing existing code and coding in the cross-platform scripting language Python.

```

#Initialization and minor procedures removed.

def is_time_to_stop_receiving():
    global buffer_full, buffer
    if not buffer_full and buffer.IsFull():
        buffer_full = 1
        return 1
    return 0

def is_time_to_resume_receiving():
    global buffer_full, buffer
    if buffer_full and buffer.HasSpareSpace(8):
        buffer_full = 0
        return 1
    return 0

while 1:
    messenger.PollSockets()

    msg = secretary.GetMsgFrom("sensor1")
    buffer.PutMsg(msg)
    msg = secretary.GetMsgFrom("sensor2")
    buffer.PutMsg(msg)

    if is_time_to_stop_receiving():
        cmd = secretary.MakeStopSendingMsg()
        messenger.SendMsg("sensor1", cmd, -1)
        messenger.SendMsg("sensor2", cmd, -1)

    if is_time_to_resume_receiving():
        cmd = secretary.MakeStartSendingMsg()
        messenger.SendMsg("sensor1", cmd, -1)
        messenger.SendMsg("sensor2", cmd, -1)

    msg2send = buffer.GetNextMsgToSend()
    if messenger.SendMsg("sink", msg2send, -1):
        buffer.RemoveSelectedMsg()

```

Figure 5-5 Implementing signaling in Python.

```

def run():
    global modify_time
    messenger.PollSockets()
    rawImage = camera.CaptureOneFrame()
    new_modify_time = os.stat("filters.py")[ST_MTIME]
    if modify_time != new_modify_time:
        modify_time = new_modify_time
        reload(filters)

    msg = filters.run(rawImage)
    buffer.PutMsg(msg)
    msgToSend = buffer.GetNextMsgToSend()
    if messenger.SendMsg("sink", msgToSend, -1):
        buffer.RemoveSelectedMsg()

```

Figure 5-6 Retasking through dynamic reloading. “filters.py” corresponds to the processing filter in Figure 5-7. It could contain the motion detection filter or not.

5.3.2 Retasking

Retasking is supported by the dynamic reloading function in Python. The code segment for retasking is shown in Figure 5-6. The Python script checks the time stamp of the filter file regularly. If a new version is available, it is loaded and executed. To enable re-tasking at different scopes, we re-organize the filters in Figure 5-3 into the structure shown in Figure 5-7. Filters in the black boxes are surrounded by the dynamic loading check and can be reloaded when changed. In this example, we are able to remove or add motion filtering and change the compression algorithm while the software is running.

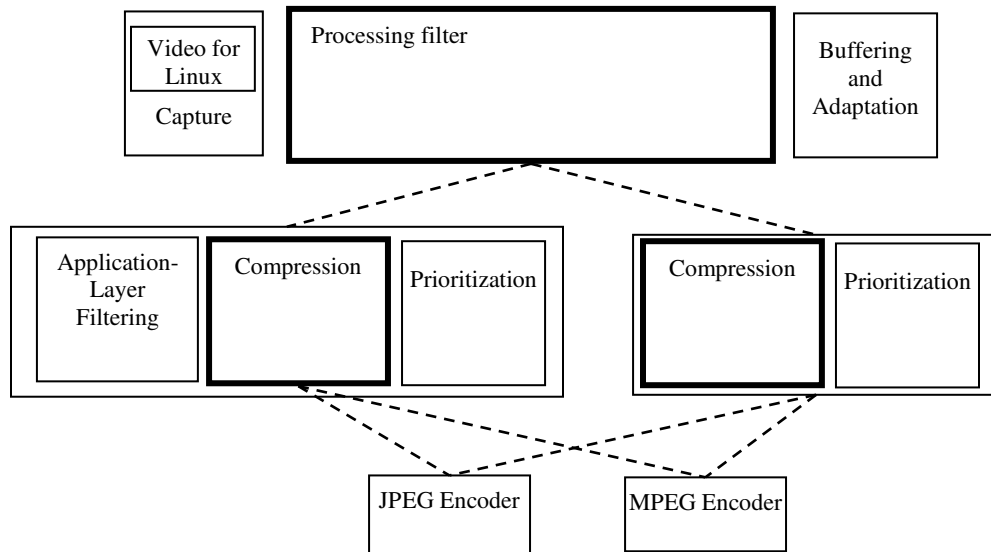


Figure 5-7 The re-organized filter structure for retasking. Black boxes are filters whose files are checked regularly and can be replaced on the fly. For example, the processing filter may include the application-layer filtering or not; the compression filter can be a JPEG encoder or an MPEG encoder.

5.3.3 Performance Experiments

To understand the impact of performance overhead on video adaptation, we have measured the amount of overhead on the *Stargate* platform introduced by connecting the system via generic interfaces in *Cascades* and the amount of extra space on the sensor needed to hold the code and Python executables.

5.3.3.1 Experimental setup

For experimentation, we compare and contrast four different types of system architecture. We have implemented a simple video collection system similar to that shown in Figure 5-3 but without an application layer filter or a prioritizer. We built the system with a single monolithic C program. We will refer to this approach as the

C approach. We have also built each of the components as standalone executables, using a shell script with pipes to interconnect the components. We will refer to this approach as the Shell approach. For the *Cascades* framework, we experimented with two approaches. Both approaches use the same compiled C modules. One approach, referred to as the Python-SWIG approach, uses the Simplified Wrapper and Interface Generator (SWIG) [74] system to generate Python interfaces for the C code. The other approach, which we refer to as Python-Native, uses hand-coded C to Python interface mappings. SWIG can generate necessary glue code automatically but may lead to excess code given its generic nature.

In the experimental set up, the sensor node is connected to a laptop through Ethernet so there is no frame dropping due to network bandwidth. Video compression is the major computationally-intensive component. We implemented three different compression algorithms because we also expect the outcome of these experiments to be useful in understanding what can and cannot be done in future multi-modal sensor networks and what the minimum preprocessing requirements are. The three compression algorithms that we implemented are JPEG, JPEG-IPP, and MPEG. The JPEG algorithm is based upon the standard libJPEG source code [38] that is freely available. The code is optimized in a CPU independent way; thus, JPEG represents a generic image compression algorithm. The JPEG-IPP algorithm takes advantage of the Intel Integrated Performance Primitives (IPP) libraries that are available from Intel[40]. The IPP libraries provide routines for copying large amount of memory, performing DCT transform, Huffman encoding, and other multimedia related tasks.

Table 5-1 JPEG performance. This table shows the performance of the libJPEG code using the four interconnect techniques. The numbers shown are frame rates in frames per second.

	C	Python-Native	Python-SWIG	Shell
160×120	29.60	29.55	29.57	27.09
320×240	10.01	10.00	9.45	8.07
640×480	2.62	2.59	2.60	2.07

The libraries are primarily low-level assembly routines that take advantage of the architecture. The MPEG algorithm is the MPEG-1 video codec from *ffmpeg*, which we optimized for the Xscale processor on the *Stargate* platform by choosing the right compile flags.

5.3.3.2 System performance

In this section, we compare and contrast the four different approaches that we have implemented: the C approach, the Shell approach, the Python-SWIG approach, and the Python-Native approach. For each approach, we captured 300 frames using each compression algorithm and measured the number of frames per second it was able to capture. The results show that the performance of the Python-based system is very close to that of the monolithic C program and better than that of the Shell-based systems.

Table 5-1 shows the results for JPEG encoding. The system is able to keep up with the camera's capture rate at the resolution 160×120 in all cases except the Shell programming case. The multiple threads and I/O necessary to move information between shell-scripted entities impose excessing overhead, as expected. Moving to

Table 5-2 JPEG IPP performance. This table shows the performance of the JPEG code that takes advantage of the IPP libraries using the four interconnect techniques. The numbers shown are frame rates in frames per second,

	C	Python-Native	Python-SWIG	Shell
160×120	29.69	29.41	29.88	28.68
320×240	18.37	18.38	17.74	13.95
640×480	5.04	5.04	5.04	3.77

320×240 frames, we see that the C, Python SWIG and Python-Native algorithms perform similarly. This is encouraging as it suggests that the overhead of using SWIG is not that high. We also notice that using shell scripting in this case requires approximately 20% overhead compared to the C approach. Finally, in the 640×480 case, we see that the processor is completely overwhelmed with data per frame. As a result all versions perform poorly while the shell scripting version is about 20% slower than the Python and C versions.

For JPEG-IPP at 160×120, as shown in Table 5-2, the *Stargate* processor is able to keep up with the camera’s capture rate. The Shell version is slightly faster than in the JPEG case primarily due to the IPP code freeing up some of the CPU cycles to do data movement and context switching between address spaces. For the 320×240 video, we see that the IPP-based code is able to achieve a video capture rate 80 to 87% better than its non-IPP-based counterpart. This suggests that in building such sensor systems, hand tuning of the filters for specific platforms is critical to performance. Meanwhile, the overhead of the Shell version increases from about 20% to 25% compared to JPEG because the computation time for each frame is decreased and the context-switch

Table 5-3 MPEG performance. This table shows the performance of the *ffmpeg* code using the four interconnect techniques. The numbers shown are frame rates in frames per second.

	C	Python-Native	Python-SWIG	Shell
160×120	22.55	21.96	21.43	20.25
320×240	8.46	8.32	8.35	7.55
640×480	2.41	2.45	2.40	2.18

overhead becomes a larger proportion of the total computation time. For the resolution 320×240 we again see that the C and various Python versions are similar. Finally, we see that in the 640×480 case, the IPP version allows nearly a doubling of the frame rate achievable by the non-IPP version.

Results for the *ffmpeg* MPEG-1 video compression algorithm are shown in Table 5-3. It is interesting to note that adding motion compensation between frames requires approximately 50% computational overhead in the 320×240 case and the 640×480 case compared to JPEG-IPP. We believe that this is partially due to the slower memory hierarchy of the embedded processor. Another point worth mentioning is that the Shell version does relatively better in the MPEG than in the JPEG cases; and the overhead decreases from 20% in JPEG to about 10%. This is due to the facts that (i) there is a significantly higher computation per frame requirement than in the JPEG cases allowing the context switching overhead to be amortized over more cycles and (ii) MPEG frames are smaller than JPEG frames on average requiring less data copying between contexts.

In general, we found that the Python-SWIG and Python-Native algorithms had very similar performance. We also found that the Python versions perform similarly to the C version. Given its interpretive nature, we believe that this is a significant achievement for the writers of Python and something that we should take advantage of for composability and retasking of sensor networking code. Finally, we note that on average the MPEG frames are approximately half the size of the JPEG frames for the 320×240 case. It may be worth spending twice the computation time (compared to JPEG-IPP) to encode video as MPEG frames to reduce bandwidth consumption in extremely bandwidth-stringent environments

5.3.3.3 Code size

One potential drawback of using Python is that it requires that the Python interpreter and necessary Python libraries (called modules in Python terminology) be installed on the each node running the Python scripts. Clearly, this could limit the types of embedded processors that the code can run on.

In Table 5-5, we have listed the code sizes for the JPEG-IPP algorithm. Here, we see the clear differences between the various approaches. The C code is a single compiled object allowing all the standard libraries to be linked in just once. For the Shell approach, the code consists of compiled code segments, which are filters, and a small script to connect them. The filters are compiled separately and each contains a copy of the standard libraries. As a result, the Shell connected code is approximately 51% larger than the C code. The Python filters (the compiled code) require even more space than the Shell filters due to the wrappers for the Python interfaces. Hand-

Table 5-5 Code Sizes. This table shows the size in kilobytes of the various subcomponents for the JPEG-IPP code.

	C	Python-Native	Python-SWIG	Shell
Compiled code	95.7	184.8	299.6	145.0
Script	--	1.0	0.6	0.6
Interpreter and libraries	--	1103.7	1103.7	--
Total	95.7	1289.5	1403.9	145.6

Table 5-4 Code Sizes. This table shows the size in kilobytes of the various subcomponents for the MPEG-based code.

	C	Python Native	Python-SWIG	Shell
Compiled code	1064.9	1072.0	1318.3	1114.1
Script	--	1.2	0.6	0.6
Interpreter and libraries	--	1103.7	1103.7	--
Total	1064.9	2176.9	2422.6	1114.7

writing the interface wrappers as in the Python-Native approach saves 120 kilobytes compared to the Python-SWIG approach, at the expense of additional programming. We also see that there is approximately one megabyte overhead to store the Python interpreter and two small Python modules that are required to run the experiments. Still, these are reasonable for all but the smallest devices.

Table 5-4, we have listed the MPEG-based code sizes. As shown in the table, MPEG requires significantly more space (and processing power) in order to operate on the embedded devices. This is as expected, given the complexity of motion estimation. Because of the relatively large size of the MPEG compiled code, much of the space

overhead of the Python interpreter and the Python modules is amortized over the larger code. For the Python-Native approach, the overhead compared to the C approach decreases from about 1250% for JPEG-IPP to 104%. For the Python-SWIG approach, the overhead decreases from 1370% to 125%. We expect that increasingly complicated video processing will be needed for adaptive video collection and will make the impact of Python's overhead smaller.

5.4 Conclusions

We believe that a component-based framework will be needed to provide programmability and retaskability to integrate multimedia data into sensor networks. We have proposed such a framework, *Cascades*, to support the processing of multimedia data in the network and the integration of multi-modal data. In this chapter, we have focused on supporting video adaptation in *Cascades*. Our experimental implementation has show that video adaptation fits well into the *Cascades* framework and has good performance.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 *Research Contributions*

In this dissertation, we have focused on addressing application-specific requirements in the process of adapting video to meet resource constraints. Existing adaptation technologies for video streaming applications may be used to meet the resource constraints; however, the mechanisms they use to lower resource consumption and to respond to network changes might not be able to deliver acceptable video for new emerging application scenarios. We have investigated new mechanisms to adapt video to meet resource constraints as well as to satisfy application requirements. In particular, we address two special requirements: accommodating large variation in resolution and collecting video in a multi-hop sensor network.

To accommodate large variation in resolution, our work focuses on supporting wide-range resolution adaptation. Our work is the first to address the problem of supporting wide-range resolution adaptation for block-based compression algorithms. We have examined the performance of existing multi-resolution video technologies when supporting a large number of resolutions; and have found the efficiency decreases rapidly as the number of resolutions increases. We have proposed hybrid

schemes and studied their performance. We have found that the *Bonneville* framework, which combines multiple scalable encodings, can make good trade-offs when organizing compressed video to support a wide range of resolutions simultaneously.

Our work on video collection in a sensor network is the first to consider adapting video in a multi-hop store-and-forward network, for non-real-time use, and for multiple video sources. We have proposed to adapt the video in the network and proposed the *Steens* framework to compose adaptation mechanisms on multiple nodes. We have designed two signaling protocols in *Steens* to coordinate multiple nodes. Our simulations show that in-network adaptation can use buffer space on intermediate nodes for adaptation and achieve better video quality than conventional network-edge adaptation. Our simulations also shown that explicit collaboration among multiple nodes through signaling can further improve video quality, reduce bandwidth wastage, and share bandwidth fairly.

We have also implemented a prototype of *Steens* on a video sensor network test-bed. The implementation is in *Cascades*, a framework we propose to support multi-modal sensor networking applications. The component-based framework of *Cascades* provides programmability and retaskability for the implementation of *Steens* while still maintains adequate performance.

6.2 Future Directions

In this dissertation, we have proposed unique video adaptation technologies to address challenges in supporting large variation in resolution and video collection in sensor networks. As with any good research, new problems have come up during our search for solutions. Below we list some of these problems that we believe are important in the area of video adaptation for future diversified video applications.

6.2.1 Future Directions in Resolution Adaptation

To accommodate large variation in resolution, we have proposed *Bonneville*, a framework for fine-grained resolution adaptation over a wide range of resolutions. However, to provide users with what they want to watch from the high resolution video shown on different display devices, we need both resolution adaptation to adjust the video to an appropriate resolution and ROI adaptation to select the right region. In *Bonneville*, video is encoded by block-based algorithms; a region can be encoded by constraining motion estimation to search within blocks that comprise that region [81][82]. The challenge is that the regions (and the resolutions) needed are not known *a priori*. We need to encode small regions and compose regions of larger sizes from them when needed. To construct regions of any size precisely, the smaller the component regions, the better. However, compression efficiency decreases as regions become smaller, since motion estimation is constrained to a region and a good match may not exist in that region. What is the compression efficiency of regional encoding, especially when combined with spatially scalable encodings, which we propose to store high-resolution video in to support resolution adaptation? Will the saving of

bandwidth by cropping irrelevant regions offset the degradation of compression efficiency? Can we plan the regions based on video content, so that large regions can be encoded that are likely to meet user needs? There are interesting questions for future work.

6.2.2 Future Directions for Video-Based Sensor Applications

There are many open questions and challenges for video-based sensor applications.

In our work, we have proposed an adaptive collection mechanism, *Steens*, that is based on priorities. We have not considered how priorities should be assigned, and assigning priorities to sensed video data is hard. Existing prioritization mechanisms usually prioritize data packets using general frame information such as the frame type and the dependencies among frames. Video collection is selective in nature so the prioritization should consider the content of the video and is very application-specific.

In *Steens*, we assume that all nodes are static and the multi-hop routes are relatively stable. Researchers [67] have proposed another collection model in which mobile “data mules” are roaming and sensor nodes send data to a data mule when it gets close. This has proven very effective for scalar sensors when sensor nodes are sparsely deployed. In the oceanographic example, the data mule could be an aircraft that flies around to collect video data. It is not clear how well this model works with video collection. Future work could consider such integration and its impact on multi-hop buffering.

Although many video sensor applications require only batched video collection, there are situations where video needs to be streamed over the sensor network for continuous playback. In this case, we still believe that collaborative in-network adaptation can do a better job than end-to-end adaptation, which most of today's adaptive streaming technologies adopt, because it can have a better estimate of network conditions and can push data dropping back, close to the source. In addition to in-network adaptation, other forms of in-network video processing, such as stitching images together to remove overlaps and filtering out irrelevant video, should be exploited to prevent potential information overload as video is streamed from multiple sensors to one destination in the sensor network. It is very challenging, though, to control the streaming latency for continuous playback when adaptation buffers and processing are both distributed.

REFERENCE

- [1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless Sensor Networks: a Survey", *Computer Networks* 38 (2002) 393-422, Elsevier Science
- [2] R. Atta, M. Ghanbari, "A Drift Compensation Architecture for DCT-Pyramid Video Coding", in *Proceedings of IEEE International Symposium on Video / Image Processing and Multimedia Communications*, Zadar, Croatia, June 2002.
- [3] B. Birney, "Intelligent Streaming", <http://www.microsoft.com/windows/windowsmedia/howto/articles/intstreaming.aspx>
- [4] V. Bhaskaran, K. Konstantinides, "Image and Video Compression Standards: Algorithms and Architectures", Kluwer Academic Publishers, Boston / Dordrecht / London, 1995.
- [5] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", *Internet RFC2309*, April 1998.
- [6] N. Bulusu, S. Jha, (editors) "Wireless Sensor Networks. A Systems Perspective", Artech House MEMS series, ISBN 1-8053-867-3.
- [7] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks", in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, Rome, Italy, July 2001.
- [8] C.-Y. Chen, R. Klette, "Image Stitching – Comparison and New Techniques", in *Proceedings of the 8th International Conference on Computer Analysis of Images and Patterns (CAIP'99)*, Ljubljana, Slovenia, September 1999.
- [9] J.-J. Chen, D. W. Lin, "Optimal Bit Allocation for Coding of Video Signals over ATM Networks", *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 6, August 1997.
- [10] M. Chu, J. E. Reich, F. Zhao, "Distributed Attention for Large Video Sensor Networks", in *Proceedings of Intelligent Distributed Surveillance Systems 2004 Seminar*, London, UK, February 2004,.

- [11] Y. Chu, S. G. Rao, S. Seshan, H. Zhang, “Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture,” in *Proceedings of SIGCOMM 2001*, San Diego, California, August 2001
- [12] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, “Video Coding for Streaming Media Delivery on the Internet”, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 11, No. 3, March 2001.
- [13] M. Domanski, L. Blaszak, S. Mackowiak, “AVC Video Coders with Spatial and Temporal Scalability”, in *Proceedings of Picture Coding Symposium*, Saint-Malo, France, April 2003.
- [14] R. Dugad, N. Ahuja, “A Fast Scheme for Image Size Change in the Compressed Domain”, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 11, No. 4, April 2001.
- [15] R. Dugad, N. Ahuja, “A Scheme for Spatial Scalability Using Non-scalable Encoders”, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 10, October 2003.
- [16] D. Estrin, L. Girod, G. Pottie, M. Srivastava, “Instrumenting the World with Wireless Sensor Networks”, in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, Salt Lake City, Utah, May 2001.
- [17] N. Feamster, D. Bansal, H. Balakrishnan, “On the Interactions between Layered Quality Adaptation and Congestion Control for Streaming Video”, in *Proceedings of 11th International Packet Video Workshop (PV2001)*, Kyongju, Korea, April 2001.
- [18] W.-C. Feng, K. G. Shin, D. D. Kandlur, D. Saha, “The BLUE Active Queue Management Algorithms”, *IEEE/ACM Transaction on Networking*, Vol. 10, No. 4, August 2002.
- [19] W.-C. Feng, B. Code, E. Kaiser, M. Shea, W. Feng, L. Bavoil, “Panoptes: A Scalable Architecture for Video Sensor Networking Applications”, in *Proceedings of ACM Multimedia 2003*, Berkeley, California, November 2003.
- [20] W.-C. Feng, M. Liu, B. Krishnaswami, A. Prabhudev, “A Priority-Based Technique for the Best-Effort Delivery of Stored Video”, in *Proceedings of Multimedia Computing and Networking (MMCN '99)*, San Jose, California, January 1999.
- [21] W.-C. Feng, J. Walpole, W. Feng, C. Pu, “Moving Towards Massively Scalable Video-Based Sensor Networks”, in *Proceedings of Workshop on New Visions for Large-Scale Networks: Research and Applications*, Washington, DC, March 2001
- [22] <http://ffmpeg.sourceforge.net/>, ffmpeg homepage.

- [23] M.L. Fisher, “The Lagrangian Relaxation Method for Solving Integer Programming Problems”, *Management Science*, Vol. 27, January 1981.
- [24] J. Foote, D. Kimber, “FlyCam: Practical Panoramic Video”, in *Proceedings of ACM Multimedia 2000*, Los Angeles, California, October-November, 2000.
- [25] H. Fujiwara, M. L. Liou, M.-T. Sun, K.-M. Yang, M. Maruyama, K. Shomura, K. Ohyama, “An all-ASIC Implementation of a Low Bit-rate Video Codec”, *IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 2, No. 2, June 1992.
- [26] D. L. Gall, “MPEG: A Video Compression Standard for Multimedia Applications”, *Communications of the ACM*, Vol.34, No.4, April 1991.
- [27] A. Gamal, L. J. Guibas, “Collaborative visual sensor networks”, <http://mediax.stanford.edu/projects/cvs.html>.
- [28] D. Ganesan, R. Govindan, S. Shenker, D. Estrin, “Highly Resilient, Energy Efficient Multipath Routing in Wireless Sensor Networks”, *Mobile Computing and Communications Review (MC2R)* Vol. 1, No. 2, 2002.
- [29] H.264/AVC Reference Software, <http://iphome.hhi.de/suehring/tml/>
- [30] B. G. Haskell, A. Puri, A. N. Netravali, “Digital Video: An Introduction to MPEG-2”, Chapman & Hall, ISBN 0-412-08411-2.
- [31] J. Heidemann, R. Govindan, “An Overview of Embedded Sensor Networks”, Information Sciences Institute Technical Report, ISI TR-2004-594, University of Southern California.
- [32] J. Hill, M. Horton, R. Kling, L. Krishnamurthy, “The Platforms Enabling Wireless Sensor Networks”, *Communications of the ACM*, Vol. 47, No.6, June 2004
- [33] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, “System Architecture Directions for Networked Sensors”, in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX 2000)*, Cambridge, Massachusetts, November 2000.
- [34] R. Holman, J. Stanley, T. Ozkan-Haller, “Applying Video Sensor Networks to Nearshore Environment Monitoring”, *Pervasive Computing*, October-December 2003.
- [35] J. Huang, W. Feng, J. Walpole, W. Jourve, “An Experimental Analysis of DCT-based Approaches for Fine-grain Multi-resolution Video”, in *Proceedings Multimedia Computing and Networking (MMCN '05)*, January 2005, San Jose, California.
- [36] J. Huang, W. Feng, N. Bulusu, W. Feng, “Cascades: Scalable, Flexible, and Composable Middleware for Multi-modal Sensor Networking Applications”,

in *Proceedings of Multimedia Computing and Networking (MMCN '06)*, January 2006, San Jose, CA.

- [37] Q. Hu, S. Panchanathan, "Image/Video Spatial Scalability in DCT Domain", *IEEE Transactions on Industrial Electronics*, Vol. 45, February 1998.
- [38] <http://www.ijpg.org>. Independent JPEG group homepage.
- [39] <http://imagemagick.org>. ImageMagick homepage.
- [40] <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>. Intel® Integrated Performance Primitives main page.
- [41] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, Massachusetts, August 2000.
- [42] S. Jacobs, A. Eleftheriadis, "Streaming Video Using Dynamic Rate Shaping and TCP Congestion Control", *Journal of Visual Communication and Image Representation*, 9(3), 1998.
- [43] S. H. Kang, A. Zakhor, "Packet Scheduling Algorithm for Wireless Video Streaming", in *Proceedings of 12th International Packet Video Workshop (PV2002)*, Pittsburgh, PA, April 2002.
- [44] C. Krasic, J. Walpole, W. Feng, "Quality-Adaptive Media Streaming by Priority Drop", in *Proceedings of Network and Operating System Support for Digital Audio and Video (NOSSDAV 2003)*, Monterey, California, June 2003.
- [45] P. Kulkarni, D. Ganesan, P. Shenoy, "The Case for Multi-tier Camera Sensor Networks", in *Proceedings of Network and Operating System Support for Digital Audio and Video (NOSSDAV 05)*, Stevenson, Washington, June 2005.
- [46] Y.-R. Lee, C.-W. Lin, C.-C. Kao, "A DCT-Domain Video Transcoder for Spatial Resolution Downconversion", in *Proceedings of International Conference on Visual Information and Information Systems (VISUAL 2002)*, Hsin Chu, Taiwan, March 2002.
- [47] Z. Lei, N. D. Georganas, "H.263 Video Transcoding for Spatial Resolution Downscaling", in *Proceedings of IEEE International Conference on Information Technology: Coding and Computing (ITCC 2002)*, Las Vegas, Nevada, April 2002.
- [48] W. Li, "Overview of Fine Granularity Scalability in MPEG-4 Video Standard", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 11, No. 3, March 2001
- [49] M. Liou, "Overview of the px64 kbit/s Video Coding", *Communications of the ACM*, Vol. 34, No. 4, April 1991

- [50] J. Lubin, "A Human Vision System Model for Objective Picture Quality Measurements", *International Broadcasting Convention*, Amsterdam, Netherlands, September 1997
- [51] S. Madden, M. Franklin, J. Hellerstein, W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", in *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, Massachusetts, California, December 2002.
- [52] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, "The Design of an Acquisitional Query Processor For Sensor Networks", in *Proceedings of SIGMOD 2003*, San Diego, California, June 2003.
- [53] S. A. Martucci, I. Sodagar, T. Chiang, Y.-Q. Zhang, "A Zerotree Wavelet Video Coder", *IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 7, No. 1, February 1997.
- [54] S. Mccanne, M. Vetterli, V. Jacobson, "Low-Complexity Video Coding for Receiver-driven Layered Multicast", *IEEE Journal on Selected Areas in Communications*, Vol. 16, No. 6, August 1997.
- [55] Z. Miao, A. Ortega, "Expected Run-time Distortion Based Scheduling for Delivery of Scalable Media", in *Proceedings of 12th International Packet Video Workshop (PV2002)*, Pittsburgh, PA, April 2002.
- [56] K. Mayer-Patel, L. Rowe, "Design and Performance of the Berkeley Continuous Media Toolkit", in *Proceedings of Multimedia Computing and Networking (MMCN'97)* San Jose, California, February 1997.
- [57] A. Ortega, "Optimal bit allocation under multiple rate constraints", in *Proceedings of the Data Compression Conference (DCC'96)*, Snowbird, Utah, April 1996.
- [58] A. Ortega, "Rate-Distortion Methods for Image and Video Compression", *IEEE Signal Processing Magazine*, November 1998.
- [59] G. J. Pottie, W. J. Kaiser, "Wireless Integrated Network Sensors", *Communication of the ACM*, Vol. 43, No. 5, May 2000.
- [60] <http://python.org>. Python homepage.
- [61] Y. Shoham, A. Gersho, "Efficient Bit Allocation for an Arbitrary Set of Quantizers", *IEEE Transaction on Acoustic, Speech, and Signal Processing*, Vol. 36, No. 9, September 1988.
- [62] K. K. Ramakrishnan, S. Folyd, "A Proposal to add Explicit Congestion Notification (ECN) to IPv6 and to TCP", Internet draft draft-kksjf-ecn-oo.txt, work in progress, November 1997.
- [63] R. Rejaie, A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," in *Proceedings of Network and Operating System Support for Digital Audio and Video (NOSSDAV 2003)*, Monterey, California, June 2003.

- [64] R. Rejaie, M. Handley, D. Estrin, "RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet", in *Proceedings of IEEE INFOCOM 1999*, New York, New York, March 1999.
- [65] A. M. Rohaly, P. Corriveau, J. Libert, A. Webster, V. Baroncini, J. Beerends, J.-L. Blin, L. Contin, T. Hamada, D. Harrison, A. Hekstra, J. Lubin, Y. Nishida, R. Nishihara, J. Pearson, A. F. Pessoa, N. Pickford, A. Schertz, M. Visca, A. Weston, S. Winkler, "Video Quality Experts Group: Current Results and Future Directions", in *Proceedings of SPIE Visual Communication and Image Processing*, Perth, Australia, June 2000.
- [66] Y. Sankarasubramaniam, O.B. Akan, I.F. Akyildiz, "ESRT: Event-to-Sink Reliable Transport for Wireless Sensor Networks", in *Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC 2003)*, Annapolis, Maryland, June 2003.
- [67] R.C. Shah, S. Roy, S. Jain, W. Brunette, "Data MULEs: modeling a three-tier architecture for sparse sensor networks", in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, Alaska, May 2003.
- [68] C.-C. Shen, C. Srisathapornphat, C. Jaikaeo, "Sensor Information Networking Architecture and Applications", *IEEE Personal Communications*. August 2001.
- [69] C. Shin, K. Seo, J. Kim, "Rectangular Region-based Selective Enhancement (RSE) for MPEG-4 Fine Granular Scalability", in *Proceedings of 12th International Packet Video Workshop (PV2002)*, Pittsburgh, PA, April 2002.
- [70] P. Sitbon, W.-C. Feng, N. Bulusu, T. Dang, "SenseTK: A Multimodal, Multimedia Sensor Networking Toolkit", in *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference (MMCN 2007)*, San Jose, CA, January 2007 (to appear).
- [71] G. J. Sullivan, T. Wiegand, "Rate-Distortion Optimization for Video Compression", *IEEE Signal Processing Magazine*, November 1998.
- [72] X. Sun, F. Wu, S. Li, W. Gao, Y. Zhang, "Seamless Switching of Scalable Video Bitstreams for Efficient Streaming", *IEEE Transaction on Multimedia*, Vol. Y, No. 2, April 2004.
- [73] X. Sun, S. Li, F. Wu, J. Shen, W. Gao, "The Improved SP Frame Coding Technique for the JVT Standard", in *Proceedings of IEEE International Conference on Image Processing (ICIP 2003)*, Barcelona, Spain, September 2003.
- [74] <http://www.swig.org>. SWIG homepage.
- [75] K. H. Tan, M. Ghanbari, "Layered Image Coding Using the DCT Pyramid", *IEEE Transactions on Image Processing*, Vol. 4, No. 4, April, 1995.

- [76] A. Vetro, C. Christopoulos, H. Sun, "Video Transcoding Architectures and Techniques: an Overview," *IEEE Signal Processing Magazine*, March 2003.
- [77] G.K. Wallace, "The JPEG Still Picture Compression Standard", *Communications of the ACM*, Vol. 34, No. 4, April 1991
- [78] C.Y. Wan, A.T. Campbell, L. Krishnamurthy, "PSFQ: a reliable transport protocol for wireless sensor networks", in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, Atlanta, Georgia, September 2002
- [79] C.-Y. Wan, S.B. Eisenman, A.T. Campbell, "CODA: Congestion Detection and Avoidance in Sensor Networks", in *Proceedings of ACM SenSys 2003*, Los Angeles, California, November 2003.
- [80] Q. Wang, Z. Xiong, F. Wu, S. Li, "Optimal Rate Allocation for Progressive Fine Granularity Scalable Video Coding", *IEEE Signal Processing Letters*, Vol. 9, No. 2, February 2002.
- [81] Y.-K. Wang, M.M. Hannuksela, M. Gabbouj, "Error-robust inter/intra mode selection using isolated regions," in *Proceedings of International Packet Video Workshop 2003 (PV2003)*, Nantes, France, April 2003.
- [82] T. Wiegand, G.J. Sullivan, G. Bjntegaard; A. Luthra, "A. Overview of the H.264/AVC video coding standard", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7. July 2003
- [83] A. Woo, T. Tong, D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks", in *Proceedings of ACM SenSys 2003*, Los Angeles, California, November, 2003,
- [84] F. Wu, S. Li, Y.-Q. Zhang, "A framework for efficient progressive fine granularity scalable video coding", *IEEE transactions on Circuits and Systems for Video Technology*, Vol. 11, No. 3, 2001.
- [85] Y. Xu, J. Heidemann, D. Estrin, "Geography-informed Energy Conservation for Ad Hoc Routing", in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, Rome, Italy, July 2001.
- [86] Y. Yao, J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks", *SIGMOD Record*, Vol. 31, No. 3, September 2002.
- [87] W. Ye, J. Heidemann, D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks", in *Proceedings of IEEE INFOCOM 2002*, New York, USA, June 2002.
- [88] P.Yin, M. Wu, B. Liu, "Video Transcoding by Reducing Spatial Resolution", in *Proceedings of IEEE International Conference on Image Processing (ICIP 2000)*, Vancouver, BC, Canada, September 2000.

- [89] Y.-Q. Zhang, S. Zafar, "Motion-compensated Wavelet Transform Coding for Color Video Compression", in *Proceedings of SPIE*, Vol. 1605, *Visual Communications and Image Processing '91: Visual Communication*, Kou-Hu Tzou; Toshio Koga, Editors, November 1991