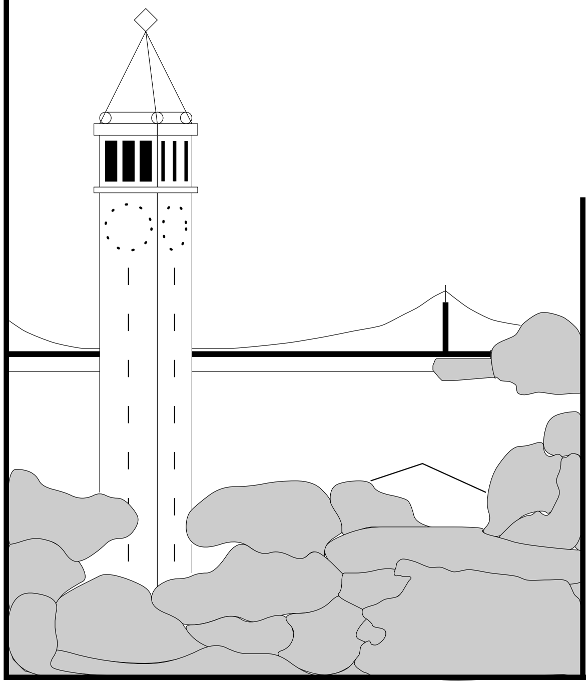
Scalable Compression and Transmission of Internet Multicast Video

Steven Ray McCanne



Report No. UCB/CSD-96-928

December 16, 1996

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Scalable Compression and Transmission of Internet Multicast Video

by

Steven Ray McCanne

B.S. (University of California, Berkeley) 1990

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Martin Vetterli, Chair
Professor Domenico Ferrari
Professor Jim Pitman

1996

Scalable Compression and Transmission of Internet Multicast Video

Copyright 1996
by
Steven Ray McCanne

Abstract

Scalable Compression and Transmission of Internet Multicast Video

by

Steven Ray McCanne
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Martin Vetterli, Chair

In just a few years the "Internet Multicast Backbone", or MBone, has risen from a small, research curiosity to a large scale and widely used communications infrastructure. A driving force behind this growth was our development of multipoint audio, video, and shared whiteboard conferencing applications that are now used daily by the large and growing MBone community. Because these real-time media are transmitted at a uniform rate to all the receivers in the network, the source must either run below the bottleneck rate or overload portions of the multicast distribution tree. In this dissertation, we propose a solution to this problem by moving the burden of rate-adaptation from the source to the receivers with a scheme we call Receiver-driven Layered Multicast, or RLM. In RLM, a source distributes a hierarchical signal by striping the constituent layers across multiple multicast groups. Receivers then adjust their reception rate by simply joining and leaving multicast groups.

But RLM solves only half of the problem. To distribute a multi-rate flow to heterogeneous receivers using RLM, the underlying signal must be encoded in a hierarchical or layered format. To this end, we developed and present herein a layered video compression algorithm which, when combined with RLM, provides a comprehensive solution for scalable multicast video transmission in heterogeneous networks. In addition to a layered representation, our coder has low-complexity (admitting an efficient software implementation) and high error resilience (admitting robust operation in loosely controlled environments like the Internet). Even with these constraints, our hybrid DCT/wavelet-based coder, which we call "Progressive Video with Hybrid transform" or PVH, exhibits good compression performance — comparable to wavelet zerotree coding (i.e., EZW) at low rates and near the performance of traditional DCT-based schemes at high rates. As well, it outperforms all (publicly available) Internet video codecs while achieving comparable run-time performance.

Our RLM/PVH framework leverages two design methodologies from two related yet often segregated fields: joint source/channel coding (JSCC) from traditional communications theory and application level framing (ALF) from computer network design. In accordance with JSCC, we combine the design of the source-coding algorithm (i.e., PVH) with the channel-coding algorithm (i.e., RLM), while in accordance with ALF, we reflect application semantics (i.e., PVH) in the design of the network protocol (i.e., RLM). In this thesis, we posit that JSCC and ALF are two manifestations of the same underlying design principle. We explore the ALF/JSCC design space with a discussion of our "Intra-H.261" video coder, which we developed specifically for MBone video transmission, and compare its performance to that of traditional designs based on independent source- and channel-coding.

Finally, we bring all of the pieces of our design together into a comprehensive system architecture realized in a flexible software toolkit that underlies our widely used video application — the UCB/LBL

video conferencing tool *vic*. Our system architecture not only integrates RLM and PVH into an autonomous video application but also provides the functionality requisite to a complete multimedia communication system, including user-interface elements and companion applications like audio and shared whiteboard. In this framework, we craft “media agents” from a common multimedia toolkit and control and configure them over a software interprocess communication bus that we call the *Coordination Bus*. By composing an arbitrary arrangement of media agents over the Coordination Bus and complementing the arrangement with an appropriate user-interface, we can induce an arbitrary multimedia collaboration style. Unlike previous work on layered video compression and transmission, we have implemented RLM, PVH, and our coordination framework in a “real” application and are deploying a fully operational system on a very large scale over the MBone.

To my wife Deana McCanne,
who gives me reason, meaning, and wholeness.

Contents

List of Figures

List of Tables

Acknowledgements

The work presented in this thesis would not have been possible without the MBone. I thank the many network researchers, too many to mention individually, who collectively made the MBone a reality. My work on the MBone tools is a synthesis of many ideas tossed about this talented research community. I further acknowledge the feedback, assistance, and contributions from the users of our tools, who put up with bug-ridden prototypes, provided detailed feedback to help diagnose problems, and contributed code for new features, devices, and machines. Again there are too many to mention individually, but I am compelled to acknowledge several individuals who contributed substantially to our efforts, including Elan Amir, Lance Berc, John Brezak, Steve Casner, Andrew Cherenson, Atanu Ghosh, Mark Handley, Jim Lowe, George Michaelson, Koji Okamura, Bob Olson, and Craig Votava.

I had the privilege of working with exceptional colleagues at the Lawrence Berkeley National Laboratory (LBNL). Kevin Fall, Sally Floyd, Van Jacobson, Craig Leres, and Vern Paxson all contributed in one way or another to my dissertation research. I owe special thanks to Van Jacobson, my supervisor at LBNL, for his creative guidance in what turned out to be a fruitful area of network research. Over the past eight years, I have learned much from Van's novel perspectives on computer networks, system architecture, and software artistry, and the work presented herein derives from our combined efforts on the MBone tools. My work at LBNL was funded by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

I thank Kathryn Crabtree, our Computer Science Division's Graduate Assistant, for fending off The Bureaucracy and for her consistent helpfulness. I also thank my peers and friends in the Tenet, Wavelet, and Daedalus Research Groups for making my graduate career not only intellectually satisfying but downright fun.

I am grateful to many of my colleagues and friends who have edited and provided feedback on this thesis (and/or papers that have contributed to this work): Elan Amir, Hari Balakrishnan, Prof. Domenico Ferrari, Sally Floyd, Ron Frederick, Amit Gupta, Fukiko Hidano, Van Jacobson, Sugih Jamin, Deana McCanne, Vern Paxson, Hoofar Razavi, Scott Shenker, Michael Speer, Thierry Turletti, Prof. Martin Vetterli, Prof. Raj Yavatkar, Prof. Hui Zhang, and Prof. Lixia Zhang. Steve Deering provided insightful comments in early discussions of RLM and first proposed that IP Multicast groups be used to distribute layered video. Ian Wakeman kindly provided me with a stand-alone reference implementation of the *ivs* congestion control scheme. I further thank Scott Shenker, Prof. Deborah Estrin, and Prof. Lixia Zhang for their recent advice on my research as well as my career path.

I am grateful to AT&T Bell Laboratories for awarding me a fellowship that funded a significant portion of my graduate education and to Silicon Graphics and Hoofar Razavi, in particular, who arranged a permanent loan of the SGI workstation that was my principal development machine. At LBNL, equipment grants were additionally provided by Sun Microsystems thanks to Michael Speer, Don Hoffman, and Allyn Romanow, and by Digital Equipment Corporation thanks to Lance Berc, Jeff Mogul, and Fred Templin.

I gratefully acknowledge Prof. Avideh Zakhor for chairing my qualifying exam committee and for insightful discussions on layered video and compression performance evaluation. I also thank Prof. Jim Pitman for serving on my committee and for offering the most challenging and most satisfying course that I took as a graduate student. I further thank Prof. Larry Rowe for many lively discussions and brainstorming sessions on multimedia networking. Despite many glitches in our software, he brought the MBone tools onto the U.C. Berkeley campus by broadcasting his weekly Multimedia and Graphics seminar beginning in the spring of 1995. I believe Prof. Rowe's ongoing experiment will ultimately be regarded as a visionary step in distance learning.

Over the course of my graduate career, I benefited invaluably from working with two stellar faculty advisors on a multifaceted research problem: Domenico Ferrari on the network side of the problem and Martin Vetterli on the compression side of the problem. I am grateful to Domenico for his insight, guidance, and support (both financial and intellectual) in the early part of my graduate career. Domenico originally proposed the video project that became *vic* and provided support for much of the project. My work in the Tenet Group was sponsored by the National Science Foundation and the Advanced Research Projects Agency (ARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives and Digital Equipment Corporation.

Martin Vetterli's constant encouragement, consistent enthusiasm, technical and theoretic savvy, and generosity with his time all combine to make him a model research advisor. I strive to emulate his ability to strike a practical balance between rigorous theoretical foundation and engineering feasibility. Despite his hiatus into the lap of Swiss luxury, we continued to collaborate while I finished my dissertation work and writing. I especially thank Martin for "showing me the ropes" of academia and for treating me as a colleague. But above all, Martin is a great friend.

Graduate school and life in general would not be the same without my close friend Elan Amir. Not only has Elan been a high caliber research collaborator, but he is a true and lifelong friend, always supportive and available through thick and thin. On the technical side, our work on the "Layered-DCT" algorithm inspired the evolution of my layered wavelet coder into the much improved hybrid wavelet/DCT coder presented in this thesis. Elan also contributed to the *vic* implementation.

Lastly, but most important, I thank my family — my parents, siblings, the Goldsmiths and O'Connors, and especially my wife Deana McCanne — who together formed the caring, supportive, and steadfast social fabric that is the sole reason for any progress I might have made. Deana has borne the brunt of my self-absorbed immersion in work and has selflessly taken on my share of countless burdens. I beg her pardon and resolve to reciprocate these favors. Above all, I am thankful for Deana's love, support, and confidence, which in the end, is all that matters.

Chapter 1

Introduction

> *I want to say a special welcome to everyone that's climbed into the Internet tonight, and has got into the MBone — and I hope it doesn't all collapse!*
> *— Mick Jagger (Nov 18, 1994)*

With these words, the Rolling Stones launched into the first audio/video broadcast of a major rock band over the Internet. Hundreds of Internet-based fans tuned in by running software-based audio/video codecs on general-purpose workstations and PCs. At the concert site, a machine digitized and compressed the analog audio and video feeds into a serial bit stream, and in turn, broke the bit stream into a sequence of discrete messages, or *packets*, for transmission over the Internet. Rather than send a copy of each packet to each user individually — as is required by the conventional *unicast* packet delivery model in the Internet — each packet was efficiently *multicast* to all receivers simultaneously using a multicast-capable portion of the Internet known as the Multicast Backbone or *MBone* [114]. Though bandwidth-efficient, this style of multipoint transmission — where a packet stream is transmitted to all receivers at a uniform rate — is undesirable because receivers are usually connected to the Internet at heterogeneous rates. For example, some users have high-speed access to the backbone, while others connect through ISDN or dial-up links. If a source's transmission rate exceeds any receiver's access link capacity, network congestion ensues, packets are discarded, and "reception quality" rapidly deteriorates. A single, fixed-rate stream cannot satisfy the conflicting requirements of a heterogeneous set of receivers, and as Jagger forewarned, large portions of the network can "collapse" under sustained congestion.

In this thesis, we address the problem posed by this Rolling Stones broadcast, namely that of delivering real-time media streams, in particular video, to heterogeneous sets of receivers connected to the network at heterogeneous rates. We believe that this style of multipoint audio/video communication over packet-switched networks will ultimately supplant the present approach to television broadcast that is now based upon analog cable networks and analog over-the-air transmission. Under this scenario, very large numbers of users will interact with the network and the system performance must *scale* with this demand. Hence, our design, analysis, and evaluation consistently addresses scalability across different performance dimensions.

A distinguishing trait of this thesis work is the particularly heavy emphasis that we placed on producing a real system that runs over real networks and is relied upon by real users to do real work. Our experiments and design work were not confined to our laboratory, but instead extended into the Internet, the MBone, and the MBone user community. Multicast use at an interesting scale requires that a large group of people spread over a broad geographic region have some reason to send and receive data to and from each other. Hence, we developed a suite of applications for multimedia conferencing over the Internet that has served as a valuable research vehicle. Our implementation-oriented approach incurred a large effort but without the design experience and feedback that results from actual deployment, it is difficult, if not impossible, to chart a path for transforming research ideas and prototypes into practical applications. Moreover, the research problems and solutions contained in this thesis have been consistently inspired by and heavily influenced by the actual use and prototyping of experimental applications.

Our overall solution for heterogeneous multicast video delivery proposes new approaches for both the network transport and compression of video, and in particular, accounts for the interdependencies between these two subsystems. Because the nature of networking stresses interaction among systems, the traditional engineering approach of analytically decomposing large problems into independent sub-problems often fails. We will later see that building a video transmission system out of existing modular components can lead to suboptimal performance. For example, a video delivery system based on analytic decomposition might result in H.261 [171] for compression and TCP [137] for transport. Although these two approaches perform well in their individual environments, their combination results in poor end-to-end performance. Instead, we synthesize an end-to-end design that accounts for component interdependencies to optimize system-level behavior.

Our solution emphasizes synthesis over analysis by adopting *joint source/channel coding* from communications theory and *application-level framing* from computer network design. Joint source/channel coding (JSCC) combines the design of compression and error-control coding to achieve better performance, while application level framing (ALF) reflects application semantics in the network protocol architecture to optimize the application for the network. By considering how the pieces of a large design interact, better design decisions can be made for each individual component. Consequently, our video communications system is based on an interdisciplinary solution that contributes to the state of the art in four key areas:

- **Signal Compression:** We designed and implemented a new low-complexity, loss tolerant, multi-rate video compression algorithm.
- **Network Protocol Architecture:** We developed a protocol framework for heterogeneous transmission of multi-rate media flows.
- **Software and Systems Engineering:** We devised a system that can be deployed in current networks and allows for incremental optimization. Our experimental compression scheme and network protocol are implemented in a "production-quality" application.
- **Networked Multimedia Tool Construction:** We devised an architecture for extensible and configurable multimedia application development where media agents are composed to create different styles of application and user-interface.

In the remainder of this introductory chapter, we motivate our work with a historical perspective of the MBone, give a high-level overview of our solution for heterogeneous multicast video delivery, identify our research contributions, and outline the rest of the dissertation.

1.1 History and Motivation: The MBone

The ultimate application of our work is to enable large scale, multicast video transmission over packet-switched networks like the Internet. Since the MBone is the enabling infrastructure for efficient multipoint packet delivery in the Internet, we first provide an overview of the MBone and related technology to motivate the research presented in later chapters.

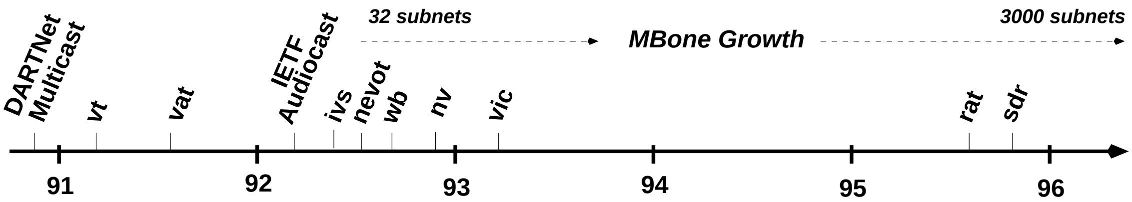


Figure 1.1: **MBone Timeline.**

Figure 1.1 depicts a timeline of the MBone, which has its roots in a collaborative project that was conceived to carry out joint network research among a number of institutions including the University of Southern California's Information Sciences Institute (ISI), the Massachusetts Institute of Technology, the Xerox Palo Alto Research Center, the Lawrence Berkeley National Laboratory, and a number of other sites. The project was sponsored by the Defense Advanced Research Projects Agency (DARPA) and a wide-area research network, called the "DARPA Research Testbed Network", or DARTNet, was created to connect the sites and serve as the principal research testbed. By 1990, the infrastructure, composed of UNIX workstations serving as programmable routers and interconnected via T1 links, was in place.

Early in the project, the research community deployed a fledgling new technology, IP Multicast [41], over DARTNet providing the first opportunity to study and experiment with wide-area network-layer multicast on a non-trivial scale. The IP Multicast architecture extends the traditional best-effort, unicast delivery model of the Internet Protocol architecture for efficient multipoint packet transmission. With this extension, group communication is very efficient because IP Multicast delivers packets to sets of receivers by forwarding packets along a spanning tree rooted at the source host. Packets are replicated only at branch points in the tree and at most one copy of each packet appears on any physical link (ignoring inefficiencies due to the tunneling mechanism required to incrementally deploy the new technology). In contrast, group communication using only unicast forces a source to "machine-gun" its packet flow to every receiver; that is, a source must send a separate copy of each packet to each interested receiver. Under this scheme, the transmission is highly inefficient because a given packet is sent multiple times over the same underlying physical links.

In support of one of its key research charters, the DARTNet community developed a number of real-time, interactive multimedia applications that exploited IP Multicast to study the problem of multiparty remote-conferencing over packet-switched networks. Building on their pioneering packet audio work from the seventies [32] and on earlier work at Bolt, Beranek, and Newman Inc., researchers at ISI crafted an audio conferencing application called the Voice Terminal, or *vt*. In February of 1991, the first packet audio conference was held over DARTNet using the Network Voice Protocol (NVP) [33] implemented within *vt*. By June, weekly research meetings among the DARTNet participants were held using NVP audio and the DARTNet multicast infrastructure.

Because *vt* was a pioneering research prototype that focused principally on network research issues, little effort went into developing its user interface. Consequently, *vt's* text-based interface was cumbersome and provided limited user control and feedback. To improve upon *vt*, we elaborated ISI's work with a graphical user interface that displays all of the participants in the audio conference and highlights the active speaker. Additionally, in joint work with Jacobson [90], we enhanced *vt's* algorithms for counteracting network packet jitter through receiver buffering and "playback point" estimation [88]. Our new application became the LBL Visual Audio Tool, *vat* [1], and the DARTNet community gradually began using our new tool upon its release in the fall of 1991[2].

The success and utility of the weekly DARTNet meetings generated interest in extending the multicast infrastructure beyond the reaches of the testbed and into other segments of the Internet. Unfortunately, at that time, production Internet routers could not carry multicast traffic. In anticipation of this problem, the IP Multicast designers enabled multicast routers to not only communicate with other routers over physical links, but also to forward packets and exchange routing messages over virtual links using IP-in-IP encapsulation[3]. In effect, a multicast router at the edge of a multicast-capable subnetwork *tunnels* through the non-multicast capable portion of the Internet to another router on an otherwise disjoint multicast-capable subnetwork.

The ability to bridge together multicast subnets using tunnels prompted an ambitious experiment in March of 1992. Thirty-two isolated multicast sites spread over four countries were configured into a large virtual multicast network, which in turn was used to *audiocast* the 23rd Internet Engineering Task Force (IETF) meeting. The virtual network backbone used to glue together the multicast-capable subnetworks was dubbed the "Multicast Backbone", or MBone [23]. For the first time, IETF participants were able to attend working group meetings and participate in the conference from a distance. Even though there were a number of technical difficulties, the experiment proved enormously successful.

This success prompted continued work on remote collaboration tools like *vat* and many new pieces came together in 1992. INRIA released their video conferencing tool, *ivs*, an integrated audio/video conferencing system that relies exclusively on H.261 [171] for video compression. In the fall of 1992, Schulzrinne released an MBone audio tool similar to *vat* called *nevot*. This tool was the principal development vehicle for the earliest versions of the Real-time Transport Protocol (RTP) [153]. RTP and the *vat* audio protocol were eventually combined, culminating in a revamped version of the RTP specification in 1995.

In the spring of 1992, we started work on a shared whiteboard application, *wb*, that augments audio and video channels with a medium for shared graphical annotation and presentation of PostScript documents. Unlike audio and video streams, which can continue in the presence of packet loss through momentary degradation in quality, *wb* requires reliable transport since drawing state is persistent and a lost drawing update should eventually be retransmitted and delivered to all of the affected participants.

The design of a reliable multicast transport protocol that scales to very large numbers of receivers in an environment like the Internet, where network partitions are common and packet loss rates are highly variable, is a fundamental challenge currently facing the network research community. In *wb*, we prototyped a new approach to reliable multicast based on transport semantics that are much more relaxed than those of traditional reliable multicast protocols. Rather than place tight constraints on message delivery order, we explicitly accounted for the whiteboard's application semantics in the design of its network protocol, relaxed the ordering guarantees, and adopted a model of *eventual consistency*. We worked out the initial *wb* software architecture and implementation for a class project [117], refined the protocol over the summer of 1992, and Jacobson improved the user interface and added functionality the subsequent fall. The underlying protocol framework, later termed Scalable Reliable Multicast (SRM), was further refined, analyzed, and simulated by Floyd et al. [59].

In December 1992, Frederick released the Xerox PARC "Network Video" tool, *nv*, a "video-only"

[1] Lacking creativity for program names, we chose the name *vat* because it was a small change from *vt*, reflecting the incremental evolution of the application. We then reverse-engineered the acronym "Visual Audio Terminal", borrowing *vt's* notion of "terminal" and Bill Joy's use of the term "visual" in the "visual editor", *vi*. Jacobson improved the name by substituting "tool" for "terminal".

[2] To ease the transition, *vat* supported NVP for backward compatibility and for some time, conferences were held using a mixture of *vt* and *vat*.

[3] IP source-routing was used originally as the mechanism for tunneling through non-multicast capable networks.

application that utilizes a custom coding scheme tailored specifically for the Internet and targeted for efficient software implementation [62]. *Nv* quickly became the de facto standard for MBone video. About the same time, Jacobson created the Session Directory tool, *sd* [86]. A user creates and advertises a conference or "session" with *sd*. In turn, each participant uses *sd* to automatically launch all the media tools pertaining to that session, freeing the user from burdensome configuration of multicast addresses, ports, and scopes. This work was refined by Handley and Jacobson into the Session Description Protocol (SDP) [75] and Handley developed a much improved SDP-based session directory tool call *sdr* [73] first released in late 1995.

In winter 1993, we started work on the UCB/LBL video conferencing application, *vic*, which became a core research vehicle for the work contained herein. We originally conceived *vic* as an application to demonstrate the Tenet real-time networking protocols [53] and to simultaneously support the evolving "Lightweight Sessions" architecture [88] that implicitly underlies all of the tools discussed. Work on *vic*[4] has since driven the evolution of the Real-time Transport Protocol (RTP) [153]. As RTP evolved, we tracked and implemented protocol changes, and fed back implementation experience to the design process. Moreover, our experience implementing the RTP payload specification for H.261 led to an improved scheme based on macroblock-level fragmentation, which resulted in a revised protocol [166]. Finally, the RTP payload specification for JPEG [10] evolved from a *vic* implementation.

The most recent major development in MBone applications is the Robust Audio Tool, *rat*, from University College London in 1995. *Rat* uses a novel forward error correction scheme where redundant information is coded at lower quality. Hence, modest packet loss rates cause minor quality degradation rather than more noticeable audio break ups.

Throughout the course of these developments, the MBone has steadily grown — from 24 subnets in the initial IETF audiocast to about 3000 subnets in mid-1996. And the content has evolved dramatically as well. No longer used exclusively for stuffy IETF meetings, the MBone has been a conduit for rock concerts, art exhibits, mainstream technical conferences, university seminars and remote teaching, dissertation defenses, NASA space shuttle broadcasts, live surgical demonstrations for medical meetings, and so forth. In just a few years the MBone has risen from a small, research curiosity to a large scale and widely used communications infrastructure.

Figure 1.2 shows a screen dump of a typical active day of MBone sessions from spring of 1995. The windows that contain lists of user names are each an instance of *vat*, while the lower right corner contains a number of *vic* application windows. Three *vic* viewing windows showing active video streams are opened on three different sessions and *wb* appears in the upper right corner. There are three active sessions: live coverage of NASA's space shuttle mission, a research meeting originating from Xerox PARC, and a seminar broadcast from U.C. Berkeley[5]. Close inspection of the video window labeled "Xerox PARC" shows a copy of *wb* running on a large in-room display device (a Xerox "LiveBoard"). Annotations to the LiveBoard appear on our local copy of *wb* as well as on everyone else's *wb*. Similarly, annotations made by any participant in the network appear everywhere, including the in-room LiveBoard.

Unfortunately, the same problem that plagued the Rolling Stones broadcast constrains other "MBone sessions" like those illustrated in the screen dump above. The heterogeneity inherent in underlying internet-

[4] The software architecture currently in *vic* and *vat* was developed essentially in the context of *vic* and retrofitted into *vat*. We originally based *vat* on the Interviews structured graphics library [113], which at the time, was one of the best solutions for user-interface construction. Ousterhout's later development of the *Tool Command Language* (Tcl) and its graphics toolkit *Tk* provided the ingredients for the much improved Tcl/C++ object architecture described in Chapter 7. In December 1993, we incorporated Tcl/Tk into *vat* based on the parallel effort in *vic*; likewise, in fall of 1995, we integrated RTP version 2 into *vat* based on the RTP architecture in *vic*.

[5] "Motion in Video Compression: What's it good for?", Michael Orchard, University of Illinois. See http://bmrc.berkeley.edu/courseware/cs298/spring95/ for more details.

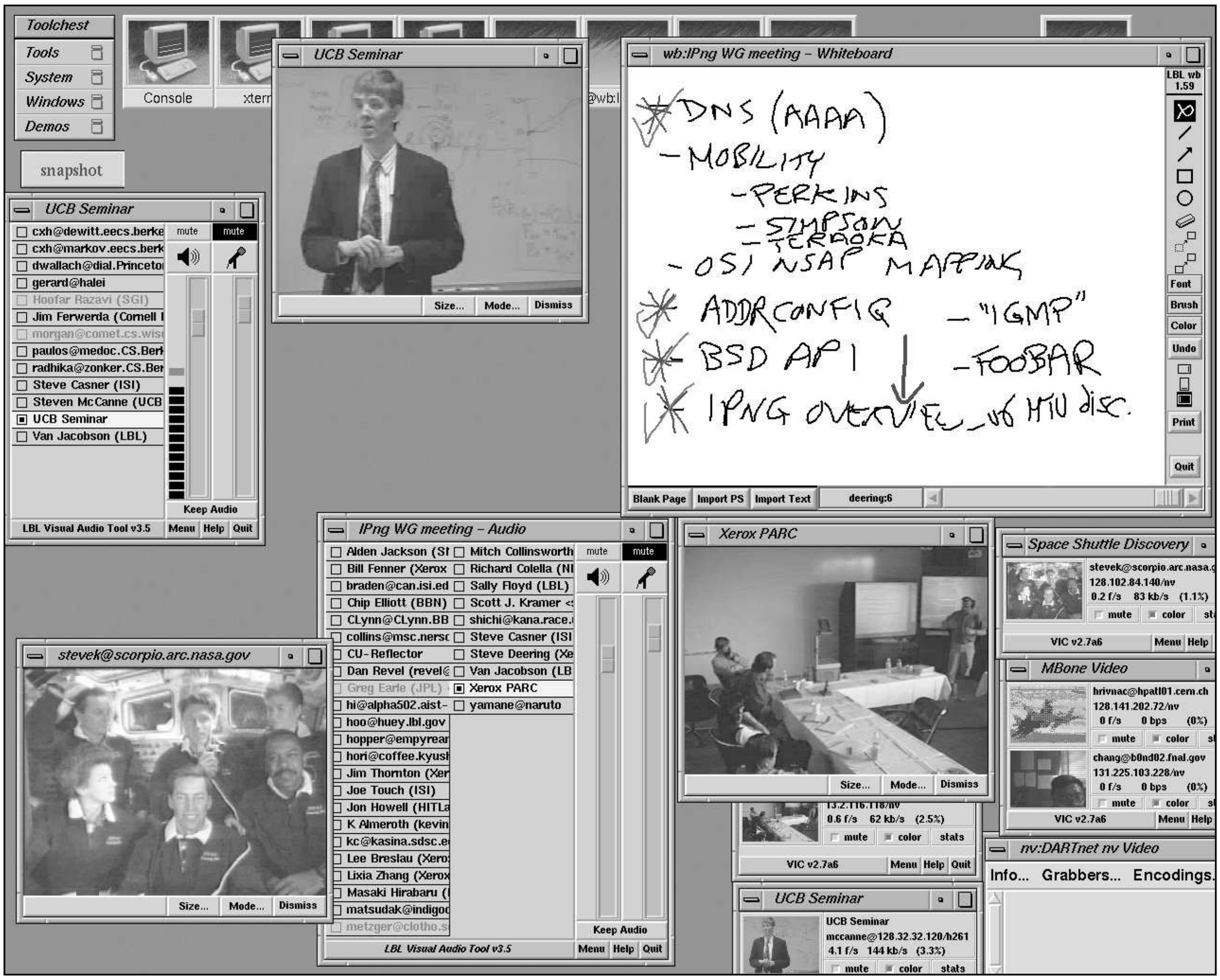


Figure 1.2: **The MBone Tools.**

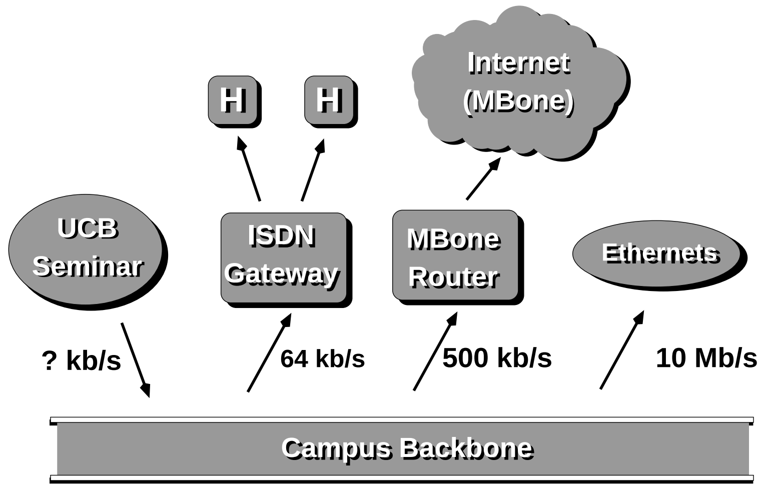


Figure 1.3: **U.C. Berkeley MBone Seminar.** U.C. Berkeley transmits a multimedia seminar over their campus network, to users at home via ISDN, and over the Internet. A single rate at the source cannot meet the conflicting bandwidth requirements of this heterogeneous set of users.

works poses difficulties in the design of a real-time media transport system. To understand this problem and its solution in more depth, we will take a closer look at the physical network topology that carries the seminar broadcasts from U.C. Berkeley.

1.2 The Problem: Network Heterogeneity

Figure 1.3 depicts the heterogeneous network topology that provides the infrastructure for the multimedia seminar broadcasts that U.C. Berkeley began transmitting in spring 1995. In this scenario, some users participate from their offices over the high-speed campus network, while other users interact over the Internet, and still others join in from home using low-rate dial-up or ISDN telephone lines. However, to maximize the quality delivered to the largest audience, Berkeley runs the transmission at a rate suitable for the MBone, which as a current rule of thumb, is 128 kb/s. But at this rate, home users cannot participate because the transmission exceeds their access bandwidth, and campus users must settle for unnecessarily low quality because the low-rate video stream underutilizes the abundant local bandwidth. If we run the broadcast at a lower rate, then users behind ISDN lines would benefit but the Internet users would experience lower quality. Likewise, if we run the transmission at a very high rate, then local users would receive improved quality, but the MBone and ISDN users would receive greatly reduced quality due to the resulting congestion. A uniform transmission rate fails to accommodate the bandwidth heterogeneity of this diverse set of receivers.

1.3 A Solution: Layered Compression and Transmission

An often cited approach for coping with receiver heterogeneity in real-time multimedia transmissions is the use of layered media streams [43, 44, 83, 122, 154, 162, 165, 173]. In this model, rather than distribute a single level of quality using a single network channel, the source distributes multiple levels of quality simultaneously across multiple network channels. In turn, each receiver individually tunes its reception rate by adjusting the number of layers that it receives. The net effect is that the signal is delivered to a heterogeneous set of receivers at different levels of quality using a heterogeneous set of rates.

To fully realize this architecture, we must solve two sub-problems: the *layered compression* problem and the *layered transmission* problem. That is, we must develop a compression scheme that allows us to generate multiple levels of quality using multiple layers simultaneously with a network delivery model that allows us to selectively deliver subsets of layers to individual receivers.

1.3.1 Layered Compression

We first precisely define the layered compression problem. Given a sequence of video frames $\{F_1, F_2 \ldots\}$ — e.g., $F_k \in [0, 255]^{640 \times 480}$ for grayscale NTSC video — we want to find an encoding E that maps a given frame into L discrete codes (i.e., into L layers):

$$E : F_k \to \{\mathcal{C}_k^1, \ldots \mathcal{C}_k^L\}$$

and further a decoding D that maps a subset of $M \leq L$ codes into a reconstructed frame, $\hat{F}_k^M$:

$$D : \{\mathcal{C}_k^1, \ldots \mathcal{C}_k^M\} \to \hat{F}_k^M$$

with the property that

$$d(F_k, \hat{F}_k^m) \geq d(F_k, \hat{F}_k^n)$$

for $0 \leq m \leq n \leq L$ and a suitably chosen metric d (e.g., mean squared error or a perceptual distortion measure). With this decomposition, an encoder can produce a set of codes that are striped across multiple network channels $\{\mathcal{N}_1, \ldots, \mathcal{N}_L\}$ by sending codes $\{\mathcal{C}_1^k, \mathcal{C}_2^k \ldots\}$ over $\mathcal{N}_k$. A receiver can then receive a subset of the flows $\{\mathcal{N}_1, \ldots, \mathcal{N}_M\}$ and reconstruct the sequence $\{\hat{F}_1^M, \hat{F}_2^M, \ldots\}$.

One approach for delivering multiple levels of quality across multiple network connections is to encode the video signal with a set of independent encoders each producing a different output rate (e.g., through controlled quantization, pixel subsampling, or frame subsampling). Hence, we can choose a $D = (D_1 \ldots D_L)$ where $D_m : \mathcal{C}_k^m \to \hat{F}_k$. This approach, often called *simulcast*, has the advantage that we can use existing codecs and/or compression algorithms as system components.

Figure 1.4 illustrates the simplicity of a simulcast coder. A video signal is duplicated across the inputs to a bank of independent encoders. These encoders each compress the signal to a different rate (and different quality). Finally, the decoder receives the signal from the corresponding encoder and decompresses the signal independently of the other layers. However, because simulcast does not exploit statistical correlations across sub-flows, its compression performance is suboptimal.

In contrast, a *layered coder* exploits correlations across sub-flows to achieve better overall compression. The input signal is compressed into a number of discrete layers, arranged in a hierarchy that provides progressive refinement. For example, if only the first layer is received, the decoder will produce the lowest quality version of the signal. If, on the other hand, the decoder receives two layers, it will combine the second layer information with the first layer to produce improved quality. Overall, the quality progressively improves with the number of layers that are received and decoded.

Figure 1.5 gives a rough sketch of the trade-off between the simulcast and layered approaches from the perspective of rate-distortion theory. Each curve traces out the distortion incurred for imperfectly coding an information source at the given rate. The distortion measures the quality degradation between the reconstructed and original signals. The ideal curve $D_I(R)$ represents the theoretical lower bound on distortion

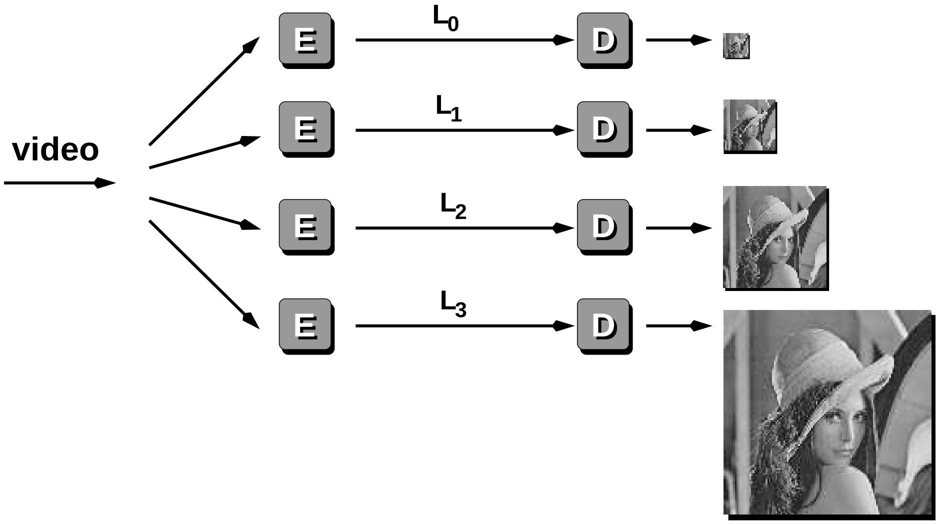


Figure 1.4: **Simulcast Video.** A simulcast codec produces a multi-rate set of signals that are independent of each other. Each layer provides improved quality but does not depend on subordinate layers. Here we show an image at multiple resolutions but the refinement can occur across other dimensions like frame rate or signal-to-noise ratio.

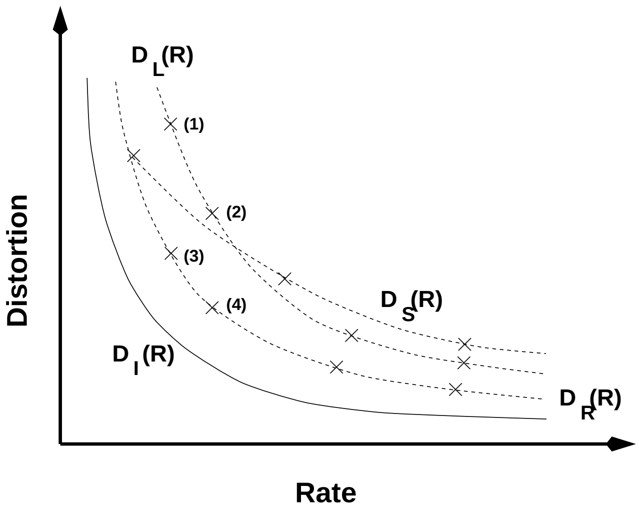


Figure 1.5: **Rate-distortion Characteristics.** Distortion rate functions for an ideal coder $D_I(R)$, a real coder $D_R(R)$, a layered coder $D_L(R)$ and a simulcast coder $D_S(R)$.

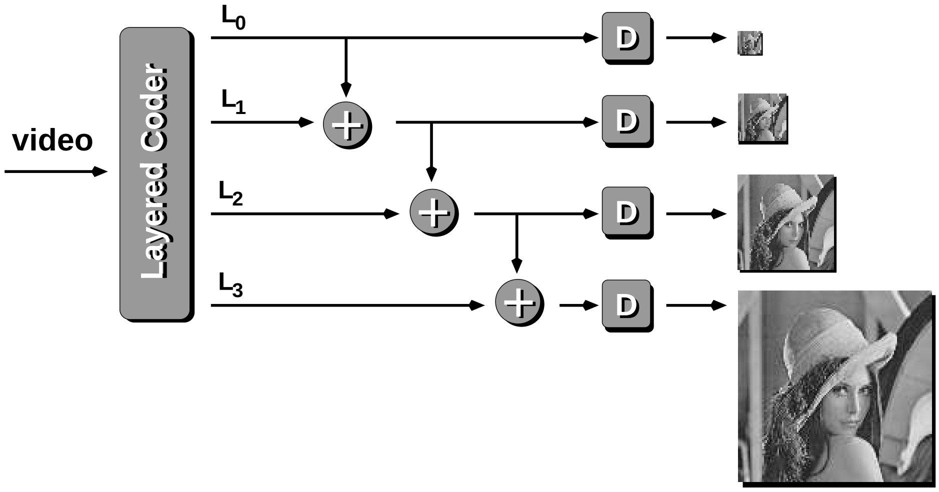


Figure 1.6: **Layered Video.** A layered codec produces a cumulative set of layers where information is combined across layers to produce progressive refinement. Here we show an image at multiple resolutions but the refinement can occur across other dimensions like frame rate or signal-to-noise ratio.

achievable as a function of rate. A real coder $D_R(R)$ can perform close to the ideal curve but never better. For example, the performance of a vector quantizer of size N approaches $D_I(R)$ as N increases [67]. A layered coder $D_L(R)$ generally performs worse than the vector quantizer because layering imposes an additional constraint [50]. On the other hand, the advantage of the layered representation is that both the encoder and decoder can travel along the distortion rate curve. That is, to move from point (1) to (2) on $D_L(R)$, the encoder carries out incremental computation and produces new output that can be appended to the previous output. Conversely, to move from point (3) to (4) on $D_R(R)$, the encoder must start from scratch and compute a completely new output string. Finally, a simulcast coder $D_S(R)$ incurs the most overhead because each operating point redundantly contains all of the operating points of lesser rate.

The conceptual structure of a layered video coder is depicted in Figure 1.6. The input video is compressed by a layered coder that produces a set of logically distinct output strings. The decoder module D is capable of decoding any cumulative set of bit strings. Each additional string produces an improvement in reconstruction quality.

1.3.2 Layered Transmission

By combining this approach of layered compression with a layered transmission system, we can solve the multicast heterogeneity problem. In this architecture, the multicast source produces a layered stream where each layer is transmitted on a different network channel, as illustrated in Figure 1.7 for the case of the UCB seminar. In turn, the network forwards only the number of layers that each physical link can support. For example, the users at home receive only the base layer across their ISDN lines, the users in the Internet receive two layers, and the users on campus receive all three. Thus each user receives the best quality signal that the network can deliver.

In this scheme, the network must be able to selectively drop layers at each bottleneck link. While

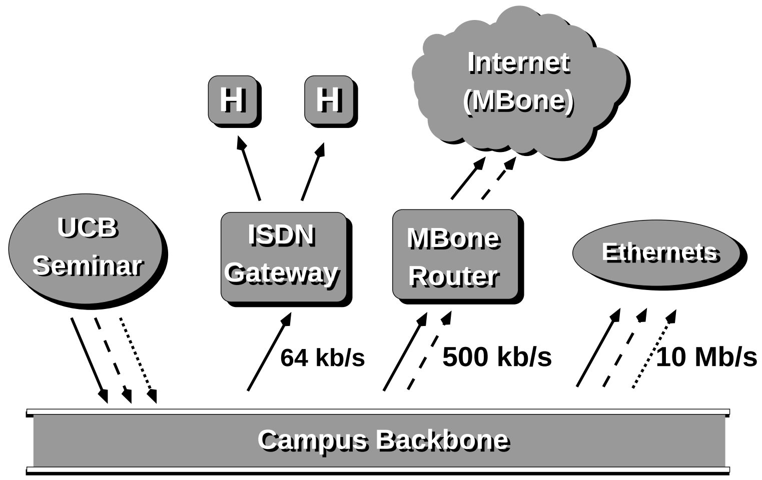


Figure 1.7: **Layered Transmission.** By combining a layered source coder with a layered transmission system, we solve the heterogeneity problem. The network forwards only the number of layers that each physical link can support.

much of the previous work leaves this problem as an implementation detail, a novel and practical scheme was proposed by Deering [43] and was further described and/or independently cited in [24, 44, 83, 122, 165]. In this approach, the layers that comprise the hierarchical signal are striped across distinct multicast groups thereby allowing receivers to adjust their reception rate by controlling the number of groups they receive. In other words, selective forwarding is implicit in receiver interest — if there are no receivers downstream of a given link in the network, the multicast routers "prune back" that portion of the distribution tree. Although this general mechanism has been discussed in the research community, an actual system based on this framework has not emerged because the problem has not been studied in detail and specific adaptation algorithms have not been developed. In this thesis, we fill this void with a specific protocol and adaptation algorithm called Receiver-driven Layered Multicast or RLM [121]. Additionally, we have designed and implemented a layered source coder based on a hybrid wavelet/DCT transform. When combined with RLM, our "Progressive Video with Hybrid-transform" codec, or PVH, provides a comprehensive solution for scalable multicast video transmission in heterogeneous networks.

1.4 Contributions

A number of research activities have laid the groundwork for both layered video compression [100, 124, 24, 162, 173] and layered transmission systems [101, 155, 43, 132, 165, 78]. However, these research efforts are each polarized: they either solve the networking half of the problem (i.e., the transmission system) or they solve the compression half of the problem. Consequently, none of these proposed systems have resulted in fully operational prototypes because, in each instance, only half of the problem is solved. Our work bridges this gap. We have developed, analyzed, simulated, and refined a comprehensive framework for layered video compression and transmission that explicitly addresses the constraints imposed by real, operational networks. We account for each component in the overall system — from the network adaptation protocol and layered compression algorithm to the application design and deployment strategy — resulting in the design and implementation of a comprehensive system for scalable multicast video distribution in heterogeneous networks.

Ultimately, we believe that broadcast-quality transmission of seminars, conferences, and the like will be commonplace on the Internet. But before this can happen, we must understand, build, and deploy scalable multicast transmission systems for real-time multimedia. This dissertation research is one step toward this goal. Our contributions advance the state of the art in packet video, congestion control, and video coding — especially in the context of best-effort multicast networks — as follows:

- **RLM.** Receiver-driven Layered Multicast is one of the first end-to-end receiver-oriented rate-adaptation algorithms for real-time multicast flows. RLM can be deployed in current networks and its performance can be incrementally optimized by adding new but "lightweight" functionality to routers. We present simulation results to show that RLM scales gracefully under several typical configurations.

- **PVH.** To complement RLM, we designed a new video compression algorithm that produces multiple layers in a progressively refinable format, tolerates packet loss, and runs efficiently in software. The scheme is based on a hybrid wavelet/DCT transform that produces a layered representation without incurring high algorithmic complexity and hence is ideal for efficient software-based implementation. Its compression performance is as good as or better than current state of the art Internet video algorithms that cannot produce a layered representation. Moreover, the run-time performance is comparable to that of existing single-layer Internet video codecs.

- **Intra-H.261.** We designed a novel coding scheme for Internet video using a fully-compliant subset of the ITU H.261 video compression standard [171]. Our scheme, called "Intra-H.261", gives significant gain in compression performance compared to the custom coding scheme in *nv* and substantial improvement in both run-time performance and packet-loss tolerance compared to the conventional H.261 algorithm in *ivs*. In addition, our development of Intra-H.261 influenced the "RTP Payload Format Specification for H.261" [166], where the fragmentation algorithm was improved through the integration of ALF[6]. Intra-H.261 is currently the most commonly used compression format for MBone video and has been incorporated into several commercial products.

- **vic.** Unlike previous work on layered video compression and transmission, we have built a fully operational system that is currently being deployed on a very large scale over the MBone. The UCB/LBL Video Conferencing tool, *vic*, was created in part as a research vehicle for this dissertation work and has become a widely used tool for MBone video conferencing. *Vic* was a critical substrate for the development of Intra-H.261, PVH, and the our system architecture for multimedia conferencing.

- **RTP.** Work on *vic* and RLM has facilitated the development and deployment of the Real-time Transport Protocol (RTP) [153]. *Vic* was the first implementation of Version 2 of RTP and was the test vehicle for many of the algorithms used in the protocol.

- **Coordination Bus.** As part of the overall work on the MBone tools, we developed a software architecture where each media is implemented as a separate tool and tools are composed via a "Coordination Bus" into different application configurations.

[6]The H.261 macroblock fragmentation scheme arose from discussions with Mark Handley and Atanu Ghosh both of University College London.

1.5 Dissertation Overview

The remainder of this dissertation is organized as follows. In the next chapter, we survey related work in the fields of packet video at large, layered compression, and Internet video applications.

Chapter 3 precisely defines the network model that we assume for all of our work. The model is based on the Internet Protocol architecture and IP Multicast service interface. We also state our assumptions for routers and argue that best-effort networks should not implement packet drop priorities.

In Chapter 4, we develop the relationship between joint source/channel coding and application level framing. We argue that these two approaches are simply different manifestations of the same underlying concept, namely, that although modular design is a powerful engineering concept, strict modularity must sometimes yield to designs that account for interaction among constituent components. We present evidence that supports these design principles and describe protocol work in Internet video and our Intra-H.261 coding scheme that is explicitly based on ALF.

Chapter 5 develops the Receiver-driven Layered Multicast protocol. We present a high-level overview of the algorithm, give details of the algorithm, and present a thorough simulation study that explores the scaling behavior of RLM. We also present simulation results that demonstrate advantages of receiver- over sender-driven adaptation.

Chapter 6 presents the layered compression algorithm PVH that complements RLM. We describe the temporal and spatial compression techniques that lead to a layered representation. We further present performance measurements to show that our layered codec runs efficiently in software, comparable to current single-layer Internet video codecs. We also describe the packetization scheme and layer resynchronization algorithm run by the receiver. Finally, we describe how PVH builds on top of RTP and present RTP extensions necessary for layered streams.

Chapter 7 presents the UCB/LBL Video Conferencing tool *vic*. We describe the software architecture underlying the MBone tools, the "Composable Tools" approach to multimedia application configuration, and the toolkit approach to the implementation of *vic* itself.

Finally, in Chapter 8, we identify a number of remaining challenges with our approach, present plans for future work, provide references to our implementation and simulation framework, and conclude.

Chapter 2

Related Work

In the early days of the Internet, experiments with digitized voice demonstrated that interactive, real-time signals could be carried over packet-switched networks [32, 33, 149]. While "packet voice" was relatively easy to deploy because voice-grade audio codecs generate fairly low-rate data streams, "packet video" was less practical because of much higher bandwidth requirements and, at the time, limitations in digital video technology. But advances in digital video compression and high-speed networking along with improved workstation performance have together made digital video applications not only feasible but also economical. These trends combine with growing public interest in the Internet and the proliferation of the World Wide Web [12] to create a new breed of multimedia-rich application requirements and new demand for packet video applications. As a result, research related to packet video has proliferated. Rather than attempt to cover the entire spectrum of packet video research, we will concentrate on the areas that are most relevant to this thesis. In the following sections, we survey related work in each of the following areas:

- packet video transmission,
- layered video compression, and
- Internet video tools.

2.1 Packet Video Transmission: An Overview

Since traditional communication networks like the telephone system are based on circuit-switched links with fixed capacities, early work on digital video transmission focused on coding video for constant bit-rate (CBR) links. Because the rate of the video signal might exceed the capacity of the underlying channel, the video transmission system must be able to adapt the rate of signal by introducing distortion in a controlled fashion. This idea — that the rate of an information source (like video) can be adjusted by degrading reconstruction quality — was born in rate-distortion theory first developed by Shannon [156]. The rate-distortion framework forms the bedrock of traditional video codec design, where codec parameters (i.e., compression quality) are dynamically adjusted to match the transmission rate of a CBR communications channel.

Figure 2.1 illustrates this model. A video signal is fed into an adjustable rate codec, where the rate is controlled via quantization parameters. The output of the codec feeds a smoothing buffer that drains at a constant rate to match the capacity of the underlying CBR channel. In turn, the smoothing buffer's instantaneous level controls the amount of quantization applied in the coding process. If the smoothing buffer fills

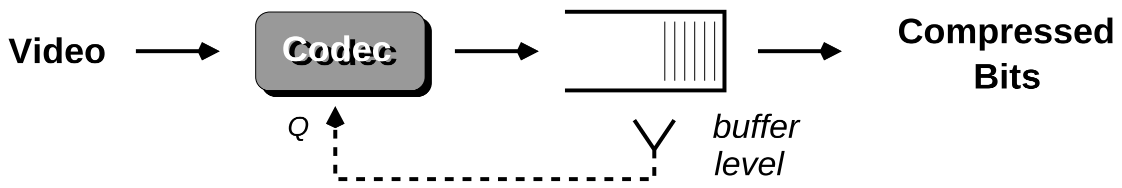


Figure 2.1: **Traditional CBR Coding Model.** In the traditional CBR coding model, the output of a variable-rate codec feeds into a smoothing buffer that drains at a constant rate. The degree of quantization is controlled by the buffer occupancy so that the buffer level is maintained at a target level to avoid buffer overflow or underflow.

too fast, quantization is increased to reduce the codec output. Likewise, if the buffer drains too fast, then the amount of quantization is reduced to prevent a buffer underflow (and a consequent stall at the decoder).

Shannon's rate-distortion framework provides a theoretical foundation for this type of buffer control strategy. If we model the video signal as a random process, then Shannon's theory provides a lower bound on the rate at which we can code a signal subject to a given level of distortion. This lower bound is related to the *entropy* rate of the process and intuitively represents the inherent compressibility of the signal. Analogous to the classical thermodynamic definition of entropy, high entropy implies that the information content of the signal is highly disordered and hence is hard to compress, while low entropy implies that the information content is highly structured or correlated and is consequently easy to compress. The entropy measure directly represents the average number of bits per symbol required to code a (discrete) information source and Shannon showed that codes can be constructed with rates arbitrarily close to the optimum.

The entropy of real video signals typically varies across the spatial extent of the image as well as from frame to frame, and this variation can be quite large. Hence, in the CBR model above, the quantization parameters must vary with time (perhaps significantly) to map the underlying dynamic entropy into a smooth output. But if the codec parameters are dynamically adjusted to match the underlying CBR transmission capacity, quality will vary significantly with time and this variation of quality has an adverse impact on perceived quality.

2.1.1 Circuit-switched Networks

Although video signals shaped for a CBR channel suffer variable quality, the CBR transmission model is well behaved, predictable, and well understood. CBR transmission has been widely deployed for many years in the *circuit-switched* telephone network. In a circuit-switched network, a "circuit" is established whenever a call is placed by allocating resources along a physical path through the network. For example, the public phone network allocates a 64 kb/s circuit for each voice call. The circuit-switched approach allows calls to be easily aggregated within the network using *time division multiplexing* (TDM). In TDM, simultaneous calls are multiplexed onto a high-capacity link by serving equal sized messages from each input stream in a synchronous round-robin fashion.

One solution to the problem of "variable quality at constant rate" is to allocate enough bandwidth in the CBR channel to handle the *peak rate* of the transmission. Although peak rate allocation allows constant quality or "fixed distortion" [39] transmission, the network is underutilized because the CBR channel is not wholly exploited whenever the variable bit-rate (VBR) signal runs below peak rate.

The efficiency of peak rate allocation is greatly reduced by the inherent burstiness of video. Chin

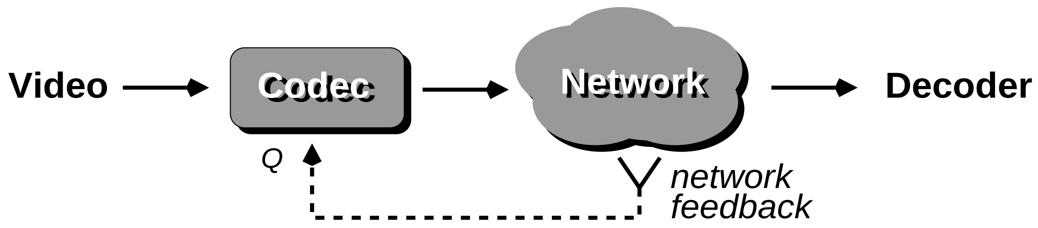


Figure 2.2: **Network-integrated Video Codec.** Gilge and Gusella's model for feedback-based congestion control of packet video. The smoothing buffer is eliminated and replaced by the network. Congestion feedback from the network, rather than the smoothing-buffer occupancy, controls the codec output rate.

et al. [28] characterized the burstiness of compressed video signals, and later, Garrett and Willenger [65] showed that typical video sources exhibit *long-range dependence*, where rate behavior is bursty even on large time scales. Consequently, there is little hope of smoothing a source into a CBR channel without large variations in quality or gross underutilization of the channel.

2.1.2 Packet-switched Networks

To maintain constant-quality transmission in the face of bursty behavior, we can replace the underlying CBR transmission model with a VBR model. In a *packet-switched* network, sources need not conform to a constant rate and multiplexing is carried out in an asynchronous fashion. Messages from multiple incoming links are multiplexed onto a single outgoing link by serving the messages in first-come first-served order. Instantaneous traffic bursts are absorbed by queuing each packet until the transmission line becomes available. As a result, messages generally experience a variable queuing delay at each multiplexor.

By using a packet-switched network, "bursty" VBR video sources can be multiplexed onto a single link giving better aggregate utilization than peak-rate allocation of CBR flows. Roughly speaking, we can multiplex a number of VBR flows so that the sum of the *average* rather than peak rates is below the link capacity. The gain realized through this type of multiplexing is called the *statistical multiplexing* gain. Unfortunately, the gain is statistical in the sense that it relies on the fact that the constituent streams will "mix nicely". That is, the bursts of each stream should rarely line up and cause the multiplexing queue to exceed its limit. When this does happen, the multiplexor is said to be *congested* and it typically resorts to discarding messages. These discarded messages, in turn, cause degraded end-to-end performance of one or more of the underlying video flows.

Not only is it possible for bursts to line up and cause congestion, but the total offered load across all video flows could exceed the fixed capacity of a bottleneck link in the network. Because video exerts a sustained, "open-loop" load on the network, as more and more flows enter the network, the network can easily become saturated and experience sustained congestion. But since a video stream can be rate-controlled by adaptively adjusting its codec parameters, all of the video sources in the network could potentially work together by adjusting their output rates to match the aggregate capacity of the underlying network. Each source could individually obtain feedback from the network and adjust its transmission rate accordingly. When the network is congested, the source decreases its rate, and when the network is underutilized, the source increases its rate. This reactive approach to *congestion control* for packet video has been proposed and analyzed by a number of researchers. We outline some of these approaches in the following section.

2.1.3 Congestion Control for Packet Video

Gilge and Gusella [70] proposed an early congestion control scheme for video transmission over best effort packet-switched networks. Their end-to-end design uses explicit feedback from the receiver to throttle the video transmission rate at the source. As illustrated in Figure 2.2, they adapted the traditional CBR coding model to packet networks by viewing the network as the smoothing buffer. The buffer occupancy feedback is replaced by loss feedback from the network.

Kanakia et al. [98] build on Gilge and Gusella's model with an architecture where the feedback signal is derived directly from the network. The bottleneck switch or router along the transmission path communicates its queuing delay back to the source. A controller uses this information to adjust the output rate of an MPEG [108] coder, allowing the source to react to queue buildup before packet loss occurs.

A very different approach is taken in the Xphone system [49], which leverages off the congestion avoidance algorithms built into TCP/IP [137]. Xphone transmits JPEG [85] video over a TCP connection and measures the throughput of the transmission. At one level, the TCP congestion control algorithm adapts the transmission rate to the available capacity in the network, while simultaneously, another control algorithm runs on top of the TCP control loop and attempts to maximize the video quality by adjusting the JPEG quantizer. In effect, the application measures the bandwidth that TCP obtains from the network and uses the measurement to adjust its source-coding rate. The measurement is input into an empirical model of an operational rate-distortion curve to update the JPEG quantizer.

Jeffay et. al. [97] view the audio/video transport process as a distributed pipeline. They designed an unreliable connection-oriented transport protocol on top of UDP/IP called the Multimedia Transport Protocol (MTP). MTP detects congestion by monitoring the local packet transmission queue. When packets are discarded due to queue overflow, the application layer is notified to reduce its data rate. This scheme works only for local area networks, where congestion results in increased media access latencies at the local adaptor.

Smith's Cyclic-UDP [159] maintains a bandwidth estimator that drives a rate-based control algorithm. At regularly spaced intervals, the receiver transmits a feedback report to the sender summarizing loss, delay, and received throughput. The sender uses a number of heuristics to adjust its estimated bandwidth based on this report. Because the target application is stored video playback, data can be buffered at both the source and receiver, allowing Cyclic-UDP to retransmit lost packets. Smith innovates over previous work by tightly coupling the media representation with the packet retransmission algorithm. Data is prioritized according to its semantic importance so that more important packets are retransmitted before less important packets.

2.1.4 Congestion Accommodation

Although congestion control algorithms limit the duration and impact of network congestion, they do not eliminate packet loss entirely. To deal with this problem, a number of schemes have been proposed to provide resilience to or recovery from short-term packet loss. One early work on packet audio proposed that short-term congestion be accommodated through the combination of layered compression at the source with prioritized packet dropping in the network [5]. Karlsson and Vetterli [100, 101] later described the integration of packet video into a general network architecture. They were the first to suggest the use of layered source-coding for video combined with prioritized packet-discard to carry out loss recovery.

In [63, 64], Garrett and Vetterli demonstrated the benefits of using network feedback to control the coding process. They were the first to describe their approach as a form of joint source and channel coding (JSCC), borrowing this terminology from traditional communication theory. In their scheme, the network uses a two-level priority scheme where routers service high-priority traffic before low-priority traffic.

Sources code some percentage, α, of their traffic as high-priority. Using feedback from the network, the sources dynamically adjust α so that the residual network capacity is allocated primarily to the high-priority traffic. While this approach optimizes the delivery of high-priority traffic in the face of congestion, the output rate of the coder is not in the feedback loop and therefore this approach by itself is not well-suited for best effort networks.

A joint source/channel coding approach is similarly taken by Ghanbari [68]. He designed a two-layer architecture, where a base layer is transmitted with guaranteed performance while an enhancement layer is transmitted on a lossy channel. His scheme is targeted for a specific slotted network which provides real-time guarantees. Similarly, Blake proposed a two-layer coder that is backward compatible with the H.261 specification [13]. He explored the performance implications of rate-control applied jointly to both layers and its impact on an admission control algorithm.

Forward error-correction (FEC) coding is often cited as a technique for coping with isolated packet loss. By adding redundant bits the transmission, a subset of the transmitted packets can reconstruct the original bit stream. The coding theory literature contains a rich array of techniques for error-correcting block codes. Yavatkar and Manoj presented some simple techniques for packet transmission based on repetition codes (i.e., packet duplication) and parity packets [178]. Although these types of error-control codes are simple to implement, they have high overhead compared to more sophisticated codes and are not widely used.

Priority Encoding Transmission (PET) [2] utilizes forward error-correction coding but considers only the so called "unequal error protection" (UEP) codes [116]. In this scheme, the source signal is partitioned according to the relative importance of different subsets of the bit stream. For example, the PET authors prioritize MPEG by separating the intra-coded (I), predicted (P), and bi-directionally predicted (B) frames into three classes of decreasing importance. The prioritized bits are encoded with the novel property that any subset M of the original N packets that comprise a message can be decoded to reconstruct the original M most important blocks of the message (with some overhead). Hence, if we assume that packets are randomly distributed among message blocks, then in the face of packet loss, PET delivers the highest priority messages possible regardless of which packets were dropped. In effect, PET provides prioritized-drop policies without explicit network support (at the cost of transmission rate overhead).

2.1.5 Integrated Services

Rather than control congestion reactively or protect against congestion with error-control coding, many researchers believe that congestion should be avoided altogether through *proactive* control, where the network is engineered so that congestion can never occur. In this approach, the network implements traffic admission controls to deny service to clients that would otherwise exceed the network's resources. A client asks for a certain type of service from the network (e.g., bandwidth, delay, and delay variance requirements) and the network runs an admission control test, which if successful, grants the client access to the requested network resources. Once the client is admitted, it produces data in accordance with its traffic descriptor. The network typically "polices" the client to prevent it from exceeding its allocation.

A network that performs call admission control and provides performance guarantees for different types of traffic is often called an "Integrated Services" network since different service classes (e.g., data, interactive multimedia, and real-time control) all share one physical network. Because of the growing interest in networked multimedia, a large body of research topical to integrated services has emerged. A comprehensive survey of all of this work is beyond the scope of this chapter and we instead describe just a few of the approaches.

A landmark work in real-time service guarantees is the Tenet Real-time Protocol Suite [53, 8]. The

Tenet project focused on providing provable performance guarantees from the network to the client. In the Tenet model, a client describes its traffic parameters and its service requirements to the network. If the network admits the client, there is an explicit contract: the network delivers the specified quality of service as long as the client adheres to its traffic description. A client may request either deterministic or statistical service guarantees.

Asynchronous Transfer Mode (ATM) networks are based on an underlying connection-oriented framework which is claimed to make admission controls and guaranteed performance more easily and efficiently implemented compared to traditional variable-length packet-switched networks. A cooperative, primarily industrial-based consortium called the ATM Forum has developed service definitions, traffic models, and protocols for providing "quality of service" guarantees in ATM networks. In [99], Karlsson surveys the integration of digital video and ATM networks.

In contrast to the focus on provable guarantees taken in Tenet and ATM, some researchers believe that "softer" guarantees can be used to simplify the network design but still provide good performance. In "Predictive Service" [30], present and past performance is used to predict future performance and hence the network can base admission control decisions on measurements rather than a priori traffic signatures [96]. This approach has the advantage that "hard" connection state need not be maintained throughout the network, but has the disadvantage that no absolute guarantees are provided.

Floyd and Jacobson proposed a mechanism rather than a service interface for network resource management called Class Based Queuing (CBQ) [58]. CBQ contrasts with traditional approaches to resource management that rely on fine-grained connection bookkeeping to provide guarantees. In CBQ, traffic is aggregated into classes and resource management state is maintained on a per-class basis. Hence, the size of the resource management state within the network scales with the number of traffic classes rather than the number of active flows. CBQ provides a flexible building block for implementing link sharing policies and has been shown to be capable of providing high-level service guarantees [54].

The ReSerVation Protocol (RSVP) working group within the Internet Engineering Task Force (IETF) is developing a protocol for establishing Internet-based integrated services reservations [179]. While RSVP defines the core setup protocol, the actual service models and infrastructure are being developed by the Integrated Services working group.

Finally, a number of researchers believe that integrated services networks can ultimately be implemented by simply overprovisioning the network. A network service provider could guarantee bandwidth and loss rates to their customers simply by building a network with enough capacity to always meet the imposed demand.

Ultimately, we believe a mixture of all these techniques will contribute to, enable, and appear in integrated services networks. Overprovisioning might be used in the network backbones, where high degrees of statistical multiplexing mean that capacity planning accurately predicts load. Likewise, resource reservations might be carried out near the access points to networks, where a small number of bursty connections could easily cause congestion. Moreover, we might combine resource allocation with application-level adaptation using CBQ. For example, a CBQ class could be allocated to handle "adaptive video among University of California campuses" and within this CBQ class, individual traffic flows could compete for capacity using adaptive rate control.

2.1.6 Encompassing Multicast

All of the previously discussed congestion control schemes for packet video are based on point-to-point video transport and do not immediately extend to multicast transmission. Comparatively little work

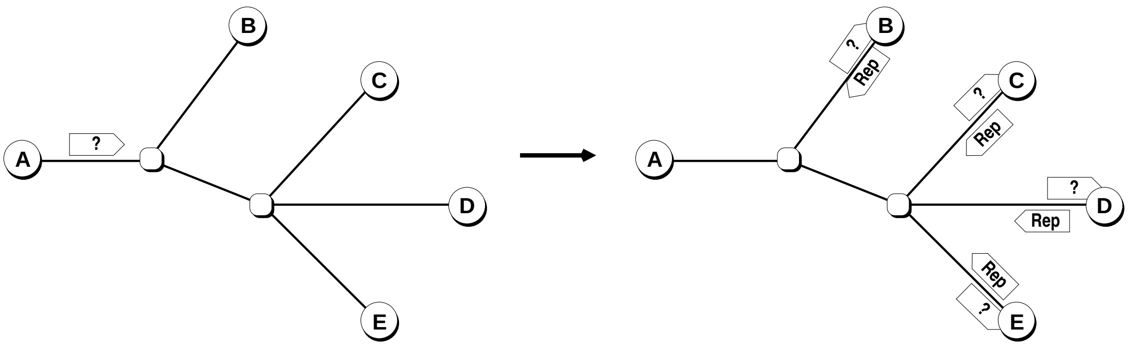


Figure 2.3: **Feedback Implosion.** Feedback implosion results when a protocol event causes all the receivers in a multicast group to synchronously generate a control message that is sent back to the source. These control messages implode at the source creating a severe traffic load that could lead to protocol instabilities and congestion collapse.

has been published on real-time multicast transport schemes but research in this area is gaining momentum.

Multicast control is fundamentally more complex than its unicast counterpart because there are multiple sources of feedback. This multiplicity of feedback causes scaling problems. In a naively designed multicast protocol, certain events could cause synchronous actions where an entire multicast group responds with feedback simultaneously. First described by Danzig [38], this *feedback implosion* can swamp the source and lead to unstable conditions. For example, a straightforward extension of unicast-based congestion control to multicast might have the source solicit feedback from all the receivers in the network. But as Figure 2.3 illustrates, this would cause feedback implosion. When the source queries the group for feedback, the entire group sends back control messages that implode at the source.

Yavatkar and Manoj [178] address the feedback implosion problem using rate-based flow-control driven by selective receiver feedback. In their Quasi-reliable Multicast Transport Protocol (QMTP), a source transmits at a constant rate over a fixed time-interval or "epoch". At the end of the epoch, the sender adjusts its rate based on feedback from receivers, and a new epoch begins. Several different heuristics are suggested for mapping the set of feedback reports into a single rate adjustment decision. Unfortunately, the suggested strategies can lead to scenarios where either portions of the network remain congested or the source is forced to run at the worst-case rate to satisfy the most constrained receiver. A critical mechanism in their protocol is the receiver feedback strategy, which must be able to scale without suffering from feedback implosion. QMTP provides two such strategies: *random delay*, where each receiver generates a feedback report at a random time chosen uniformly across the epoch, or *probabilistic backoff*, where upon detecting loss, a receiver generates a feedback report with a fixed probability. While these two approaches increase the scalability of the protocol, they do not eliminate the implosion problem altogether.

The IETF's Audio/Video Transport Working Group (AVT) has standardized the "Real-time Transport Protocol" [153], or RTP, for real-time multimedia delivery over multicast (and unicast) networks. RTCP, the control protocol embedded in RTP, requires that receivers transmit *reception reports* back to the multicast group, which contain loss, throughput, and delay statistics as seen by the reporting host. Rather than unicast the control messages back to the source, they are multicast to the group to provide better scaling properties. Since each receiver sees all other reports, the time interval between reports can be dynamically computed so that the aggregate report bandwidth is some small percentage of the real-time media bandwidth.

The RTP specification suggests that the reception reports could be used in a feedback control loop where the source would adjust its transmission rate based on the state of the network [153, Section 6.3.4]. Busse et al. [20] first implemented this approach (and worked out many unspecified details) by modifying *vic* [120] with a specific instance of an RTCP-based multicast feedback control algorithm. Their experiments show that for a very simple configuration (two sources each sending to a single receiver), the algorithm adapts to changing loss rates and tracks the available bandwidth, but they do not explore the scaling behavior of the algorithm. One drawback with this approach is that as the number of receivers in the group grows, the reception report interval is decreased, which in turn, delays the feedback signal. Delayed feedback at best increases the duration of congestion periods and at worst causes controller instability, as it is well known from control theory that system stability is sensitive to delay in the feedback loop.

The most widely deployed reactive congestion control scheme is a feedback protocol developed by Bolot, Turletti and Wakeman [15]. Their protocol is implemented in the INRIA Videoconferenceing System (*ivs*), which like *nv* and *vic*, runs over RTP and IP Multicast. As in related schemes, they use feedback from the network to control the output rate of the video coder, in this case, an H.261 [171] software codec. Their protocol employs a novel probing mechanism to elicit feedback from the network that is based on a probabilistic solicitation of the receivers to determine the size of the multicast group and the worst case state of congestion.

Unfortunately, each of these source-based rate-adaptation schemes — QMTP, RTCP-based feedback, and *ivs* — is poorly matched to multicast environments. Each approach is fundamentally incapable of coping with network heterogeneity since conflicting feedback information must be mapped into a single rate-adjustment decision. Either low-capacity regions of the distribution are overwhelmed or high-capacity regions are underutilized. As described in Chapter 1, any scheme that relies on adjusting the rate of the transmission at the source cannot simultaneously meet the conflicting requirements of a heterogeneous set of receivers.

2.1.7 Network-assisted Bandwidth Adaptation

Rather than adjust the transmission rate at the source, some researchers have proposed that the rate of a media flow be adjusted from within the network. In this way, heterogeneity is "locally" accommodated by fine-tuning the transmission rate to exactly match the available capacity on each link in the network.

In [132, 133], Pasquale et al. cite the shortcomings of closed-loop congestion control for multicast networks. Their Multimedia Multicast Channel (MMC) is designed specifically for heterogeneity by imposing only a loose coupling between the receivers and sources. In their architecture, receivers connect to a multicast channel through a port. By attaching a "filter" to the port, a receiver instructs the network to transform a flow to a lower rate. In turn, the network optimizes the delivery of heterogeneous flows by propagating the receiver-specified filters up the multicast distribution tree. At merge points in the tree, filters of the same type are combined and further propagated up the tree. While this architecture supports heterogeneous transmission, it does not provide a mechanism for end-systems to adapt dynamically to the available capacity in the network. Instead, MMC relies on admission control to explicitly allocate available bandwidth to heterogeneous receivers.

The *CU-SeeMe* system [47] from Cornell University distributes video over a manually configured multicast distribution mechanism. So called *reflectors* are manually placed throughout the network to effect a multicast distribution tree using native unicast transmission. Receivers generate loss reports that are sent back up the distribution tree via the reflectors. Hence, the reflectors can collapse loss reports to avoid the implosion problem. But they do so by averaging statistics across each reporting host. Because the sending rate

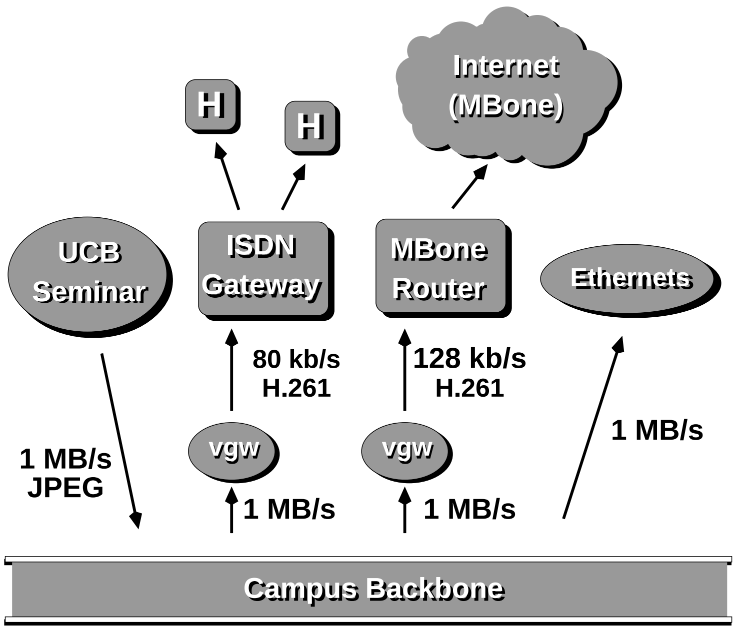


Figure 2.4: **Network-assisted Adaptation.** In network-assisted bandwidth adaptation, the rate of a video transmission is adjusted within the network. For the UCB Seminar transmissions, we transmit high-rate JPEG streams over the campus network and use multiple copies of our video gateway *vgw*, placed at strategic locations in the network, to transcode the high-rate video into low-rate H.261 for the MBone and for home users across ISDN lines.

is based on an average over different receivers, the overall transmission quality is proportional to the worst-case receiver. Moreover, the manual placement of reflectors works poorly in practice because end users are rarely capable of deploying the reflectors at strategic locations not only because of administrative access restrictions but also for lack of knowledge of the underlying topology. Consequently, "reflector networks" are most often deployed by creating a centralized server to which each user connects in a star topology. Typically, a user creates a "conference" by setting up a reflector and advertising its Internet host address. Each participant in turn connects to this advertised address, placing not only a computational load on the reflector host but also a traffic load on the network near the reflector. In this configuration, the server replicates each incoming packet to each of the outgoing connections causing quadratic growth of traffic.

The RTP architecture explicitly allows application-level gateways to carry out format conversion and rate-adaptation between "RTP sessions". Turletti and Bolot [165] describe an architecture based on this model where *video gateways* are placed throughout the network to transcode portions of the multicast distribution tree into a lower bit rate coding, using either the same coding format with different parameters or a different coding scheme altogether. This video gateway architecture was first fully developed and fielded by Amir et al. [4] in a specific application called *vgw*, which provided design feedback for the RTP specification and contributed an efficient software-based method for transcoding high-rate Motion-JPEG video into low-rate H.261. For example, network heterogeneity underlying the UCB MBone seminar transmissions described in Chapter 1 has been handled with video gateways. As illustrated in Figure 2.4, gateways are deployed at strategic locations within the campus network to adapt a high-rate video transmission for the

MBone and ISDN. A 1 Mb/s Motion-JPEG stream is transmitted over the campus network and two gateways are deployed. One gateway transcodes the high-rate stream to a 128 kb/s H.261 stream for the Internet, while the other produces an 80 kb/s stream for users over ISDN.

Tennenhouse and Wetherall's "Active Network Architecture"[163] provides a vastly generalized approach for the deployment of rate-adaptive video gateways within the network. In their architecture, network nodes carry out arbitrary computation to implement not only network functions but also user-specified algorithms. Hence, video gateways can be deployed by placing them at arbitrary points in the network.

2.1.8 Receiver-based Schemes

Although all of the schemes based on network-assisted bandwidth adaptation solve the heterogeneity problem, they place both an administrative as well as a computational burden on the network. Moreover, the approach requires new network technology that is not currently in place and, depending on the approach, could be difficult to deploy incrementally. Although in some cases it may be possible to deploy transcoding gateways at strategic locations in the network, an architecture that requires transcoding at arbitrary, bottleneck links is enormously burdened. Nodes internal to the network, where resources are most critical, must perform intensive computational tasks and be capable of processing arbitrary coding algorithms. Even if these problems are overcome, the transcoding process fundamentally increases end-to-end delay, and furthermore, cannot be applied to a secure communication without entrusting the network with the encryption key.

To avoid the problems inherent in the gateway architecture and in the source-based adaptation schemes, we and others have proposed that the burden of rate-adaption be placed neither on the source nor on the network but rather on the receivers. Using the layered compression and transmission model described in Chapter 1, receivers adapt to local capacity in the network by adjusting their level of subscription to the transmission.

The earliest appearance in the literature of the layered compression/transmission model for multicast was due to Shacham [155]. In his Heterogeneous Multicast (HMC), network and end-system heterogeneity is handled through a combination of layered source coding and layered packet forwarding. This early work focused on optimal route computations that maximize the aggregate delivered rate across all of the destinations for a given traffic mix. The connection-oriented model assumes that each layer is allocated resources as part of an admission control procedure.

The HMC route optimization architecture has several shortcomings. First, because HMC is based on the notion of link-state routing — that is, the full network topology and traffic load are known to the route computation module — it scales poorly. Each time an end-system adjusts its requirements (or a new receiver joins or leaves the group), routes must be recomputed and in a large network, this can occur frequently. Second, small local changes can dramatically impact the global optimization, precluding an incremental optimization structured in a distributed, scalable fashion. For example, if one receiver drops a layer, then the entire distribution topology might need to be re-routed (see [123, Appendix] for a proof that optimality requires a global exchange of information). We believe that theoretic optimality should be sacrificed in favor of local and scalable computation. Third, in order to compute the optimization, the routing algorithm must know all of the source rates ahead of time and each layer must be constant-rate. For many applications, source rates cannot be known a priori. Finally, the framework assumes the existence of an end-to-end infrastructure for resource-controlled multicast connections, which is not (and may not ever be) universally deployed.

The "Discrete Scaling" mechanism in the Heidelberg Transport System (HeiTS) [44] uses a receiver-oriented scheme for adapting to delivered bandwidth. Here, receivers open and close ST-II [22] multicast connections to adapt to bandwidth. The authors do not discuss adaptation algorithms or report implementation results.

As described in Chapter 1, Deering first suggested that the IP Multicast be used as a layered transmission system where layers are individually mapped onto multicast groups [43]. Both Turletti and Bolot [165] and Chaddha and Gupta [25] describe this architecture but do not present an adaptation algorithm or implementation. Our approach was first described in [118] and [122], but a specific adaptation scheme was not published until [121]. Brown et al. have implemented a multi-resolution extension to the *CU-SeeMe* video conferencing system where IP Multicast receivers subscribe to either a 160x120 or a 320x240 stream by joining either one or two multicast groups [18]. Receivers drop down to the 160x120 resolution when they detect high packet loss rates. The RSVP architecture was designed to explicitly accommodate layered flows using resource reservations [179]. Similarly, Heffner proposed the use of layered media streams in tandem with resource reservations [80] in the context of the Tenet Scheme 2 protocol suite [71].

Concurrent with our work, Hoffman and Speer built a system based on the layered multicast architecture [83]. They use multiple frame rates of JPEG video to generate a temporal hierarchy and employ two techniques for adaptation. Their first technique is a negotiation algorithm run by each receiver that obtains the highest available quality of service explicitly from the network (e.g., using RSVP). Their second approach uses layered multicast with an aggressive adaptation scheme where a new receiver subscribes to all the layers in the distribution and drops layers until the quality of the delivered stream is adequate. Although this approach might perform adequately on small scales, it will not handle large sessions or dynamic bandwidth fluctuations because each new receiver causes maximal disruption to the rest of the session by initially joining all of the groups. Also, there is no mechanism for discovering bandwidth when it becomes available. Since the only action is to drop a layer, over time, each receiver eventually drops down to the minimal level of subscription.

Campbell and Coulson proposed a receiver-oriented video delivery system that relies on layered compression [21]. Their "QoS adaptor" assumes a native ATM network that provides performance guarantees on a base-layer stream. Receivers issue "reservation" signaling messages to the network that reflect the level of congestion. In the presence of congestion, the receiver reduces the resource request in the reservation. ATM switch managers coalesce reservation messages and send them up the distribution tree toward the source in a fashion reminiscent of Pasquale's filter propagation [132]. The network informs the source of the new requirements, which in turn may cause the source to alter its transmission rates. If so, an "adapt" message is broadcast to the receivers to update their knowledge of available resources.

Cheung et al. [27] extended the *ivs* congestion control scheme with "destination set grouping" (DSG). Under DSG, a source transmits its signal at multiple rates across a set of multicast groups. The receivers are partitioned across the groups and each group's rate is controlled via the *ivs* algorithm. In addition, receivers adaptively move among groups according to packet loss rates. Although their system is based on simulcasted MPEG streams, they describe a hypothetical architecture based on layered video and resource reservation.

The DSG authors presented measurements of their system across the Internet, but the experiments were of limited scale and we conjecture that the protocol will suffer adverse performance for large sessions. Because group changes are initiated by the source and receivers use a non-adaptive decision process, the protocol does not learn from past congestion to avoid future congestion. More specifically, if a receiver R is in group G_1 and experiences no congestion, it subsequently moves to the next destination-set group, which we call G_2. If R then encounters congestion, it reverts to G_1 and repeats the process at fixed intervals (i.e., the control algorithm resonates rather than decays). Moreover, before returning to G_1, R causes the *ivs* control loop in G_2 to back off to its lower limit, which may take a significant amount of time. Unfortunately, since the source continually transmits packets during the G_2 excursion, the path to R becomes severely congested for a sustained period of time.

The DSG architecture represents a hybrid of the source-based and receiver-based congestion control methods. Delgrossi et al. describe a similar hybrid using HeiTS [44] where they suggest that their "continuous" (i.e., source-based) and "discrete" (i.e., receiver-based) scaling methods could in principle be combined in a single approach, but they did not develop or implement this idea.

2.1.9 Summary of Packet Video Work

In the previous sections, we summarized related work in a number of areas relevant to the network transport issues in this thesis, namely:

- early packet video work,
- source-based rate-adaptive congestion control,
- network-assisted rate adaptation via gateways, and
- receiver-based rate adaptation via layered transmission.

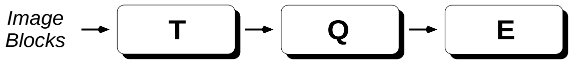
Although these research efforts provide a rich spectrum of work to build on, none of these contributions have completely solved the problem of multicast delivery in heterogeneous, connectionless networks. The early work on packet video does not address multicast delivery, and the source-based congestion control schemes emphasize solutions to the implosion problem rather than the overall goal of maximizing end-to-end delivered quality across *all* receivers. Source-based schemes control only one variable, the transmission rate, so all receivers are forced to receive the same level of quality. The rate must be chosen as a compromise between the high- and low-bandwidth paths, and how this compromise is carried out is arbitrary policy. Both QMTP and the *ivs* scheme propose such policies, and the compromises they must make are inherently unsatisfying. The "active network" adaptation schemes place undue burden on the network and cannot be easily deployed in an incremental fashion. Finally, the existing work on receiver-based protocols either lacks specific algorithms for adaptation, relies on connection-oriented resource reservations, or does not scale beyond small numbers of receivers.

2.2 Layered Video Compression: An Overview

While techniques for transmission of video over networks are critical to an overall packet video delivery system, equally important components include the compression and coding algorithms for the video signal itself. Rather than cover the entire range of video compression techniques, we survey the work most relevant to this thesis, i.e., *layered* video compression. As we explain in later chapters, no existing video compression algorithm meets all of the requirements of our system. Hence, we developed a new codec, optimized for our environment, that leverages off many of the algorithms and techniques presented in the following sections.

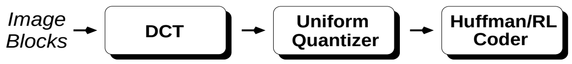
2.2.1 Layered DCT

Image and video compression algorithms based on the Discrete Cosine Transform (DCT) have been enormously successful and are now used in a number of international image and video coding standards, e.g., the Motion Picture Experts Group (MPEG) [108] video compression standard and the Joint Photographic Experts Group (JPEG) [85, 135] image compression standard. The DCT-based algorithms are instances of a generic "transform coder", which is decomposed as follows:



In this model, a transform, T, is applied to the input image to "decorrelate" the pixel values. If the transform optimally decorrelates the signal, then we can apply a quantizer, Q, to each coefficient individually without loss of efficiency. The transform representation is additionally beneficial because, in practice, perceptual biases are more easily exploited in the transform domain than in the pixel domain (e.g., high frequency DCT coefficients are typically quantized more coarsely than low frequency coefficients). The quantized coefficients are then compacted through entropy coding, E. Note that the transform and entropy-coding stages are both reversible and the quantization stage is solely responsible for introducing controlled distortion — and hence increased compressibility — into the algorithm.

A well known property of the DCT is that it approximates the optimal decorrelating transform, the Karhunen-Loeve transform, for continuous tone images with high spatial correlation [141]. As a result, standards like MPEG and JPEG have adopted the DCT as the transform component in the diagram above:



Here, 8x8 image blocks are first decomposed using a DCT. These coefficients are then quantized with a uniform scalar quantizer. Finally, runs of zeros are run-length coded and the run-codes and quantization levels are combined into Huffman codes [84].

This is the basic algorithm used by JPEG. A color image is represented by three bit-planes (a luminance plane and two color planes) and broken up into a number of 8x8 blocks. Each block of each plane is fed into into the pipeline above (in a well defined scan order) and the output of the entropy stage is an aggregate code that represents the image. This codeword can be fed into the inverse system and "decoded" to yield a reconstructed version of the original. The compression performance of the overall system is typically characterized by considering the distortion incurred for different levels of compression[1].

Unfortunately, the codeword that results from the JPEG process is not a successively refinable, or *layered*, code. We cannot take a prefix, or any obvious subset of the bits, and produce a representation of the original with a gracefully degraded quality. If we took a prefix of a standard JPEG code, we would simply get the first N blocks of the image in scan order rather than the entire image at lower resolution or lower spatial quality.

A number of techniques have been developed to produce compressed layered representations using DCT-based coding techniques. The techniques fall into roughly three categories:

[1] Distortion is computed as a distance metric between the original and reconstructed images. There is a wealth of emerging literature on the problem of finding a good metric that reflects perceptual distance as induced by the human visual system, but the most commonly cited metric is the so called peak signal-to-noise ratio (PSNR):

$$d(X, \hat{X}) = 10 \log(M^2 / \sum_{1}^{N} (X(i) - \hat{X}(i))^2)$$

where M is the maximum value of each sample, $X(i)$ is the original signal, and $\hat{X}(i)$ is the reconstructed signal. In later chapters, we use this metric to compare and evaluate the compression performance of different algorithms.

- spectral separation,
- successive quantization, and
- spatial scaling.

In each case, the image is processed in a number of scans. In spectral separation, each scan encodes a different set of related DCT coefficients, e.g., the more important low-frequency coefficients are coded before the high-frequency coefficients. In successive quantization, the DCT coefficients are successively refined by decreasing the granularity of the quantization step size on each scan. For example, decreasing the quantization interval by half on each scan is equivalent to sending an extra bit of precision of each coefficient, i.e., sending the coefficients a bit-plane at a time. Finally, in spatial scaling, the spatial resolution of the image is increased at each stage of decoding. In the first stage, a coarse scale image is decoded. This image is then upsampled, filtered, and used as the prediction for the next stage.

These three schemes all appear in the JPEG standard. While spectral separation and spatial scaling are rarely used, the successive quantization algorithm has been widely used because the Independent JPEG Group has publicly distributed a robust implementation. They utilized the point-transform technique in the JPEG standard [85, Annex G] to produce a layered JPEG codec, which is now referred to as "Progressive-JPEG". Even though Progressive-JPEG is constrained to produce a layered representation, it often outperforms the non-layered baseline JPEG algorithm. Using the Independent JPEG Group's implementation, we found that Progressive-JPEG outperforms baseline JPEG by by 0.5 to 1dB of *peak signal to noise ratio* (PSNR) for the 512x512 Lena test image [3].

Progressive-JPEG is now widely deployed in the Internet because vendors of World Wide Web browsers embraced the technology soon after it was implemented and made freely available by the Independent JPEG group. In a Web browser, the key advantage of a layered format like Progressive-JPEG is not multi-rate transmission but rather serial progressive refinement of an image transfer. If a user accesses the Web across a low-bandwidth link, a large image transfer takes many seconds. But when represented in Progressive-JPEG format, the image appears after only a short time, albeit at coarse scale. But gradually, details "fill in". At any time, the user can move on and abort the current image transfer thereby increasing application interactivity.

Unfortunately, the layered representation of Progressive-JPEG results in a run-time overhead greater than its single-layer counterpart. In the case of the Web, bottlenecks in network transmission far outweigh run-time performance bottlenecks and performance is of little concern. But for real-time video processing, the performance of the standard Progressive-JPEG algorithm degrades with the number of layers encoded. To reduce this effect, Amir et al. [3] designed a variant of Progressive-JPEG, called "Layered-DCT" (LDCT), specifically for Internet video coding. LDCT runs significantly faster than Progressive-JPEG with equivalent or better compression gain.

Like JPEG, the MPEG-2 standard has provisions for hierarchical bit streams, where a signal is decomposed into two or more bit streams that represent different resolutions, quality (i.e., PSNR), or frame rates. Several researchers have additionally suggested using the I, P, B frame structure of MPEG to induce a temporal hierarchy [155, 2, 44].

2.2.2 Pyramid Coding

A landmark work in multiresolution image coding is Burt and Adelson's pyramid coding framework [19]. The basic structure is illustrated in Figure 2.5. The input image is first downsampled and low-pass

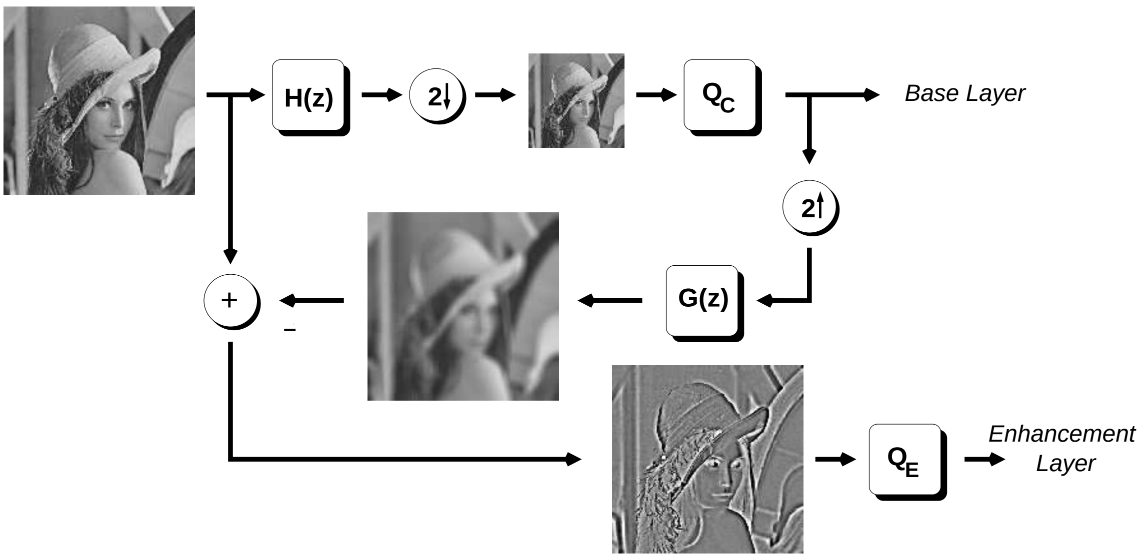


Figure 2.5: **Laplacian Pyramid Coding Structure.** This diagram illustrates one stage of a Laplacian Pyramid coding structure. The input signal is low-pass filtered and downsampled to give a coarse scale representation, which is quantized by Q_C and transmitted on a base-layer channel. The coarsely quantized signal is upsampled and interpolated to form a prediction signal, which is quantized by Q_E and transmitted on the enhancement channel.

filtered to create a coarse scale representation. The coarse scale image is quantized and coded. Because this coarse scale representation contains much less information than the original image, it can be coded in much fewer bits, but at the cost of decreased quality. The novelty of the pyramid structure is that the original quality can be recovered by incrementally refining the low quality representation. To do so, the encoder uses the coarse scale image to predict the original: the coarse image is upsampled, filtered, and subtracted from the original signal to produce a prediction error. Since the spectral fingerprint of natural scenes tends to have energy concentrated in the low frequencies, this prediction error (which accounts for high-frequency errors in the low-pass coarse image) typically has low energy and is "easy" to code.

The net result is a two-layer hierarchy consisting of the coarse scale signal that is transmitted on a base-layer channel and the prediction error that is quantized independently and transmitted on a separate enhancement-layer channel. Rather than settle for just two levels of hierarchy, Burt and Adelson extended the structure by recursively applying the two-step decomposition to the coarse-scale signal. In this fashion, an arbitrary number of refinement layers can be computed by transforming the base-layer image to coarser and coarser quality, forming the so called "pyramid" structure. This type of pyramid coder is often called a "Laplacian Pyramid" because the refinement bands are equivalently obtained by convolving the original image with an appropriately scaled Laplacian weighting function.

The Laplacian pyramid was a break-through innovation that led to a number of image and video coding schemes. One example is Chaddha et al.'s scheme [26, 24] based on a Laplacian pyramid and tree-structured vector quantization [67] of the refinement bands. Both the vector codebook construction algorithm and vector searches during the encoding process are computationally expensive, but the decoder algorithm can be carried out efficiently almost exclusively with table lookups. Thus, their scheme is well-suited for

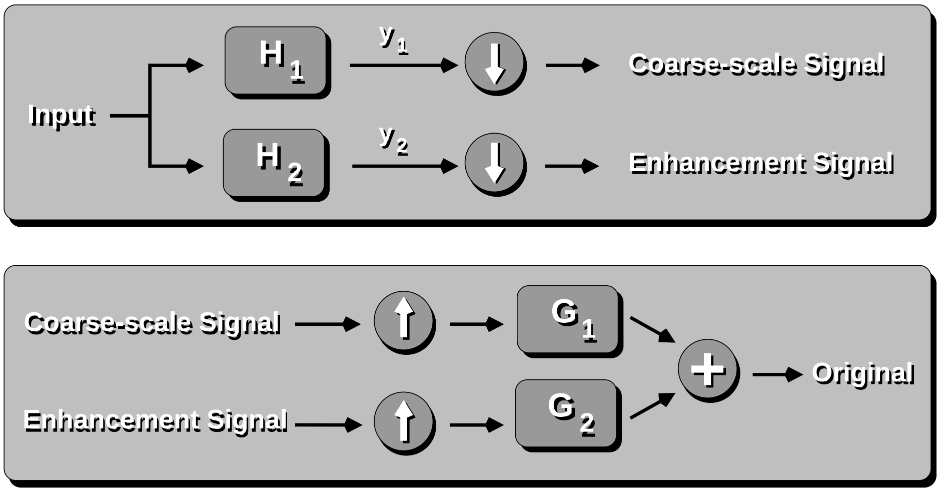


Figure 2.6: **Subband Decomposition Primitives.** The fundamental operation in subband analysis splits the input signal into a coarse-scale and an enhancement signal, each half the size of the input, via filtering and downsampling. With "perfect reconstruction" filter pairs, the original signal can be recovered by upsampling, filtering, and merging the two subsignals.

stored video where the encoding process can be carried out off-line but the decoding process is interactive, real-time, and cheap.

2.2.3 Subband Image Coding

A marked disadvantage of the Laplacian pyramid coding structure is the expansion of signal samples caused by the prediction step. That is, a 512x512 is decomposed into a 256x256 coarse-scale block plus a 512x512 refinement block, incurring 25% overhead. In the limit, the overhead is bounded by $1 + 1/4 + 1/16 \ldots = 1/(1 - 1/4) = 4/3$ or 33%, but this is a significant cost. In typical coding schemes, a 33% reduction in bit rate results in an obvious degradation in quality. While one could argue that this overhead is inconsequential if the pyramid structure admits a compression scheme that overcomes this disadvantage, in practice such performance has not been realized. Instead, schemes based on *critically subsampled* signal expansions have yielded better compression performance.

One such scheme is based on subband signal decomposition using perfect (or near-perfect) reconstruction subband filter pairs. The fundamental operation in a subband analysis system is depicted in Figure 2.6. A signal is split into two subsignals, each with half as many samples as the original, through filtering and downsampling. As in the Laplacian pyramid, the signal is decomposed into a coarse-scale version and an enhancement version, but in this case, the decomposition is carried out with a pair of filters — a low-pass filter $H_1(z)$ and a high-pass filter $H_2(z)$ — usually called the *analysis* filters. The resulting two subsignals $y_1(n)$ and $y_2(n)$ are then quantized, coded, and transmitted independently. To reconstruct the original signal, y_1 and y_2 are upsampled, filtered by a pair of *synthesis* filters $G_1(z)$ and $G_2(z)$, and added together.

For appropriately designed subband filter pairs, the original signal is recovered exactly, assuming no quantization. But to provide compression, the intermediate signals y_1 and y_2 are usually quantized and entropy coded resulting in degradation in the reconstructed signal. However, the subband filter structure gracefully accommodates the "noise" introduced by the quantization process. Moreover, if the enhancement signal is absent, we can set $y_2(n) = 0$ at the input to $G_2(z)$ to recover the coarse-scale version of the original signal (i.e., quality degrades gracefully).

A number of theoretical results have been obtained that lead to efficient algorithms based on subband decomposition. One key result is that certain classes of filters, namely orthonormal filters, preserve the L_2 norm across analysis and synthesis. Accordingly, distortion introduced by the quantization of subband coefficients is reflected back onto the original signal representation in a well defined manner. That is, inducing 1dB of distortion through subband coefficient quantization induces exactly 1dB of distortion to the reconstructed signal. As a result, efficient "bit-allocation algorithms" — algorithms that decide how much of the rate budget to devote to different pieces of the coding algorithm — can be developed that operate entirely within the subband domain and produce well defined, potentially "optimal", results in the pixel domain.

In practice, the advantages of preservation of the L_2 norm are outweighed by the advantages of other aspects of filter design. The filters used in many practical subband coding systems are neither orthonormal nor perfect-reconstruction. Stopband transitions, phase response characteristics, and ringing effects turn out to have a larger impact on the coding and perceptual performance of many practical algorithms. For example, the filters described in [1] perform well in practice but provide only *nearly* perfect reconstruction. However, the errors introduced by reconstruction imperfection are much smaller than those due to quantization and compression.

As with the Laplacian pyramid, we have described a two-level subband decomposition that can be recursively applied to the coarse scale subband and iterated an arbitrary number of times. This process leads to a hierarchy of subbands each half the size of the original. Although we developed this decomposition from the viewpoint of filtering and downsampling, an alternate interpretation is that of a linear transformation. Amidst a flurry of research activity in the 1980s, an equivalence was established between subband filtering and discrete signal expansions using wavelet transforms. In the transform interpretation, the input signal is mapped onto a new basis by computing the set of corresponding basis coefficients in the same way a signal is decomposed into coefficients of complex exponentials using the Fourier transform. For this reason, we often interchange the terms "subband filtering", "subband transform", and "wavelet transform" throughout this dissertation.

The simplest instance of a perfect-reconstruction, orthonormal, subband filter pair is the Haar filter pair:

$$G_1(z) = (1 + z^{-1})/\sqrt{2} \tag{2.1}$$
$$G_2(z) = (1 - z^{-1})/\sqrt{2} \tag{2.2}$$

Intuitively, these filters generate a two-tap average signal and a two-tap difference signal. When iterated, the Haar filters perform signal expansion onto a square wave basis [72]. Because of this, quantization noise typically shows up as obvious "blocking artifacts". Moreover, the Haar filter has poor energy compaction because of its gradual stopband transition.

Much research has been carried out that explores tradeoffs in filter design for subband analysis, but most of this work does not address the issue of implementation complexity. One exception is LeGall's work on a low-complexity video coding system [107] that uses the following biorthogonal filter pair:

$$\begin{aligned} H_0(z) &= -1 + 3z^{-1} + 3z^{-2} - z^{-3} \\ H_1(z) &= -1 + 3z^{-1} - 3z^{-2} + z^{-3} \end{aligned}$$

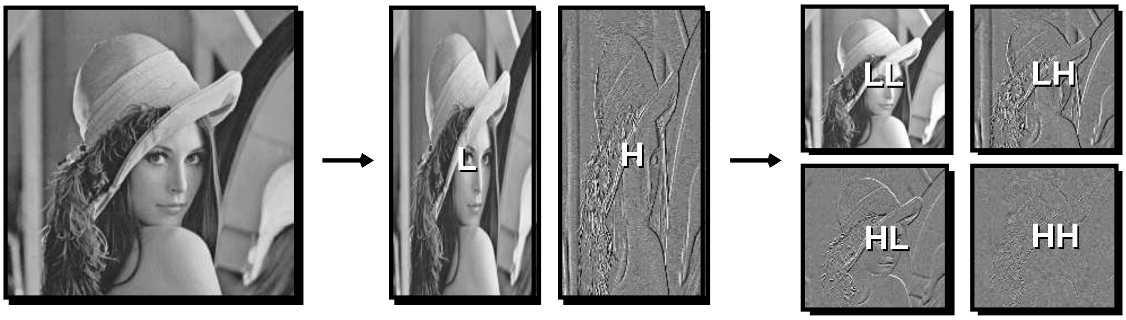


Figure 2.7: **Two-dimensional Wavelet Decomposition.** The canonical wavelet decomposition as applied to images. A four-level analysis is carried out by first applying filtering and downsampling horizontally, resulting in two subbands each half the original size. The process is repeated vertically over each intermediate subband yielding the four-level decomposition on the right.

with the following synthesis

$$\begin{aligned} G_0(z) &= (1 + 3z^{-1} + 3z^{-2} + z^{-3})/16 \\ G_1(z) &= (-1 - 3z^{-1} + 3z^{-2} + z^{-3})/16 \end{aligned}$$

The basis functions that underly this subband transform are much smoother than the Haar basis functions so that quantization artifacts have a smaller perceptual impact. Moreover, the filters can be efficiently computed in software (and hardware) using only shifts and adds.

Vetterli first suggested that wavelet transforms be applied to image coding by applying separable subband filters sequentially across the horizontal and vertical directions [168]. An example of this widely used separable 2D subband/wavelet decomposition is shown in Figure 2.7. We first apply the subband analysis filter pairs horizontally, which results in the coarse scale and enhancement subsignals often called the L subband and the H subband. Note that because of the downsampling operations, the overall number of signal samples has not changed. We then apply the vertical filters to each of the new subbands. This results in the four band decomposition consisting of a coarse scale LL band and the LH, HL, and HH enhancement subbands. Woods independently developed this coding model and designed the first compression schemes based on it [177].

A layered structure is explicit in this representation and one obvious approach for a layered coding algorithm is to code each subband independently. A three-layered signal could be generated by sending the LL band on one channel, the LH and HL on another, and the HH band on the last channel. Naturally, we might want more layers, and as described above, we can recursively apply this four-stage decomposition to the LL subband. The hierarchy that results from this recursive application is shown in Figure 2.8.

Subband Coding Methods

All of the subband analysis techniques described thus far simply transform the signal representation and do not compress it. The power behind the wavelet representation is that it admits efficient techniques for perceptually-based compression algorithms because its multiscale structure provides a hierarchy that complements the "resolution adaptability" of the human visual system.

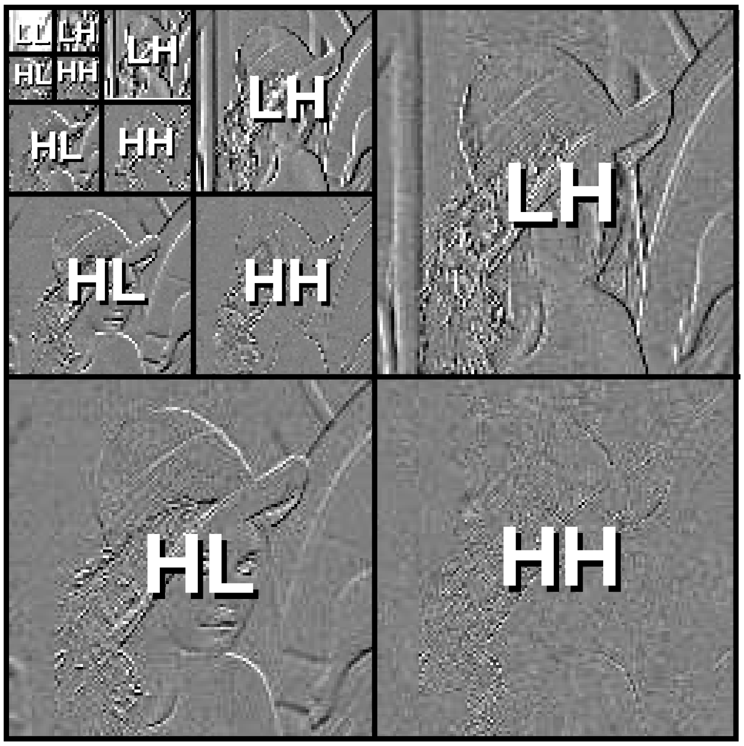


Figure 2.8: **Hierarchical Wavelet Decomposition.** A hierarchy of subbands is generated by recursively applying the four-stage decomposition from Figure 2.7 to the LL subband.

One of the simplest subband coding techniques is to process the subbands in some well defined "scan order" and code each coefficient of each subband independently [169]. A bit-allocation optimization determines how many bits of precision to allocate to each band (e.g., the coarse scale bands are typically "most important" and are allocated more bits per coefficient than the finest scale subbands). Each coefficient in the enhancement subbands is coded independently using an entropy code tailored to the statistics of the quantized coefficients. The coarse scale coefficients are generally correlated and coding them differentially can improve compression performance.

Cross-subband Coding Schemes

Although one of the goals of subband analysis is to decompose an image into a decorrelated set of subbands by isolating the details of the image at different levels of scale, in practice subband representations of natural images have strong intersubband correlations. One intuition for this effect is simply that there exists a natural correlation at spatially localized regions across scales for real objects, e.g., an interesting scene at one scale often has interesting detail at finer scales. A stronger effect is that a sharp, "high energy" edge can require contributions from many wavelet bases all in a spatially localized region of the image. Thus, coefficients corresponding to the same localized region will be correlated across subband scale (i.e., from LH_0 to LH_1 and so on) and orientation (i.e., from LH_1 to HL_1 to HH_1).

Lewis and Knowles dramatically improved subband coding techniques by exploiting these intersubband correlations in their image compression algorithm [111]. They arrange subband coefficients in the now well-known "quadtree" structure. For each coefficient (except those in the LL and the "leaf" subbands), we identify the four children coefficients whose basis vectors are centered at the spatial location of their parent. In turn, we identify the children of each child and so forth. This arrangement leads to a structure in which each of the coarsest scale enhancement subbands contain a forest of trees where each tree corresponds to the set of basis vector coefficients that are centered over their respective location in the coarse scale subband.

The coefficients are then scanned one tree at a time and each tree is traversed in depth-first order. If a coefficient falls below a heuristic threshold (that attempts to model perceptual factors), the entire tree beyond that point is compactly coded with a single symbol that represents "all zeros". However, this approach suffers from two key problems. First, it attempts to predict insignificance across scales without explicitly exploring the signal energy in the tree below the point deemed to be all zeros. Consequently, a high-energy edge that appears only at fine scales would be missed. Second, although the coder is based on subband decomposition, it does not immediately admit a layered representation.

Shapiro solved both of these problems in a scheme called "Embedded Zerotrees of Wavelet coefficients" (EZW) [157]. He improved upon Lewis and Knowles' scheme by creating a fully embedded code through successive refinement of quantization intervals. When the quantization intervals are refined by powers of two, the scheme is equivalent to bit-plane coding of the subband coefficient magnitudes. One of Shapiro's key innovations was to isolate the coding of the dominant bits (i.e., the most significant non-zero bit of the coefficient) from the refinement bits (i.e., the bit positions with magnitude less than the dominant position) and the "isolated zero" bits (i.e., the zeros to the left of the dominant bit). He observed that the entropy of the refinement bits is more or less one bit per bit and consequently that almost all of the compression gain is exploited through efficient coding of the location and position of the dominant bits. He recast Lewis and Knowles depth-first scanning algorithm into a more sophisticated scan over the bit planes and coefficient quadtree and termed the "all zeros" subtree a *zerotree*. By carefully separating the coding of the zerotrees from the refinement bits, he achieved much better compression performance because the scheme better captures correlations across subband scales. Intuitively, Lewis and Knowles' scheme fails to capture much of the intersubband correlation because refinement bits add “noise” to the coding structure that identifies dominant bits.

The EZW output code has the enormously useful property that it is an *embedded* code. In an embedded code, any prefix of a valid codeword is likewise a valid codeword and if the code is chosen appropriately, the prefix codeword corresponds to a lower-quality version of the longer codeword. Hence we can trivially form a layered code from an embedded code by breaking up the longer code at arbitrary boundaries. Moreover we can tune the rates of the different layers since we have the freedom to break the embedded code at an arbitrary point.

Said and Pearlman [146] improve EZW by using a two-level zerotree data structure. They describe their algorithm in terms of “set partitioning” of the wavelet coefficients, which is an alternate interpretation of the EZW algorithm. In turn, they effectively extend the notion of zerotree with a symbol that we call a “grandparent zerotree”. Unlike a zerotree root, where the coefficients below that point are all zero, a grandparent zerotree root implies that all of the coefficients two generations below and beyond are zero and the children may or may not be zero. By introducing the grandparent zerotree coding symbol, Said and Pearlman achieved impressive compression performance gains, and by casting the algorithm as a set-partitioning problem and eliminating the arithmetic coding stage, they produced the fastest known (software-based) zerotree-based image coder.

2.2.4 Subband Video Coding

All of the techniques cited thus far are examples of two dimensional image coding algorithms and do not immediately apply to video. A trivial extension of subband image coding to video is to code each frame of the video sequence independently, as in “Motion-JPEG” where a sequence of pictures are independently coded using the JPEG image compression standard. But this results in poor compression performance because the scheme does not exploit the tremendous frame-to-frame or “temporal” redundancies in typical video sequences.

Spatio-temporal Wavelet Pyramid. Karlsson and Vetterli were the first to extend subband coding techniques to video by introducing spatio-temporal wavelet pyramid [101]. In this model, subband decomposition is carried out not only spatially across each frame but also temporally from frame to frame. The simplest example of a temporal subband filter is the Haar wavelet. In this case, two frames are coded at a time. Rather than code each frame separately, the Haar decomposition creates two new frames that represent the average and difference of the two original frames. Hence if there is little scene activity, the difference frame is mostly zero and compresses well. Otherwise, if there is high motion, the frame-average captures the motion as “blur” and the frame-difference contains the enhancement information required to recover the original detail. Clearly, this scheme can be generalized to an arbitrary number of frames by using longer and/or iterated temporal subband filters.

3D Subband Video Coding. Taubman and Zakhor [162] build on both the spatio-temporal pyramid structure and the intersubband coding techniques with a layered video compression algorithm that performs on par with the best non-layered schemes. Their video codec exploits intersubband correlations in a fashion similar to EZW but they constrain their coefficient scan to maintain the spatial multiresolution property, i.e., substreams can be extracted from the embedded video code to produce a specific range of video resolutions.

A fundamental difficulty with temporal subband coding is that temporal filtering does not model motion well. Despite many attempts, the ability of temporal subband coding algorithms to exploit motion is consistently inferior to the traditional block-based motion-compensation techniques that MPEG and H.261

utilize. Consequently, some researchers have proposed coding systems based on motion compensation where the motion prediction error is coded through subband decomposition [69]. Unfortunately, all of these schemes entail high implementation complexity and fail to outperform traditional schemes based on motion vectors and the DCT.

WWHVQ. One of the first works from the compression community to specifically target the problem of Internet multicast video delivery is Vishwanath and Chou's "perceptually-Weighted Wavelets using Hierarchical Vector Quantization" (WWHVQ) [173]. Their hybrid scheme combines hierarchical vector quantization with wavelet decomposition. The approach is especially attractive because it can be implemented efficiently in software using only table lookups. However, their scheme does not produce an embedded code and hence requires active manipulation to transcode the bit rate.

WWHVQ exploits temporal redundancies through conditional replenishment of subband coefficients, i.e., only coefficients that change beyond a fixed threshold from frame to frame are coded. Although the subband replenishment technique works well when the analysis is based on Haar filters, it is problematic when used with longer filters. The Haar filter is the only filter for which the transform's basis vectors are non-overlapping. Hence after iterating a Haar filter, a given coefficient quadtree corresponds exactly to a localized block of pixels in the original image. However, iteration of longer filters leads to basis vectors that spread out over large regions of the image. Usually this is a desirable property (because quantization results in spatially diffuse degradation), but it hampers the coefficient replenishment algorithm. We have found that small changes in fine-scale coefficients can cause dramatic changes in large sets of coefficients across the image. For example, background noise in an otherwise static video scene can cause phase inversion of large numbers of subband coefficients. Consequently, we believe that the WWHVQ coefficient replenishment scheme is effective only in tandem with Haar filters, but further study is required to categorically resolve this issue.

2.2.5 Summary of Layered Video Work

The work on layered video compression presented in the previous section is far from comprehensive. A large number of layered coding schemes have been proposed, including schemes based on progressive DCT [68, 142, 104], subband coding [39, 11, 158], and pyramidal coding [139]. We reviewed some key areas in layered compression research and cited representative or landmark work in each area. Although there is a wealth of existing work in layered compression on which to build, none of this work meets the overall systems-oriented constraints of our layered video system and hence we cannot utilize "off the shelf" technology. Instead, as described in later chapters, we exploit existing technology as system sub-components and weave these pieces into a novel configuration that is optimized for our layered transmission environment.

2.3 Internet Video: An Overview

One of the key contributions of this thesis is the Internet video conferencing application *vic* along with its underlying software and system architecture. Not only has *vic* served as the research vehicle for this dissertation work, but it is also a novel contribution in its own right. In this section, we survey related work on applications, systems, and architectures for video transmission over the Internet.

2.3.1 Xerox PARC Network Video

One of the earliest widely-used applications for Internet video transmission is the Network Video tool *nv* from Xerox PARC [62]. An important prototype in the development of the early RTP draft standards, *nv* is based on open-loop transmission of real-time video to an arbitrary number of receivers using IP Multicast. Rather than implement all conferencing media in a single application, *nv* is a “video-only” application that relies on an external application for the audio channel (or for any other media).

A particularly novel aspect of *nv* is its custom coding scheme based on a Haar wavelet decomposition. The compression algorithm is tailored specifically for the Internet and targeted for efficient software implementation [62]. Moreover, the compression and packetization schemes were designed jointly and as a result, *nv* video streams exhibited much better end-to-end performance compared to other techniques that at the time were based on existing coding models. Despite its use of a wavelet decomposition, the *nv* compression scheme does not have a layered representation.

In addition to its custom coding format, *nv* eventually supported a range of video coding formats and adopted an extensible architecture that facilitated experimentation with new formats. Sun’s CellB format and the *CU-SeeMe* compression formats were both incorporated into the application. The *CU-SeeMe* codec enabled the development of a *CU-SeeMe* reflector-to-MBone bridge that allowed *CU-SeeMe* streams to be forwarded across the MBone and decoded and viewed using *nv*.

The network model assumed by *nv* is very simple. It relies on the consistent availability of adequate bandwidth and does not have any form of reactive congestion control. Instead, the user configures a rate-limiter with the desired transmission bandwidth and the network is expected to deliver that rate of video. Although in general this approach could cause serious congestion problems, *nv* was used successfully over the MBone through careful engineering of bandwidth and transmission scopes and tightly coordinated use of the MBone through public mailing lists. Though this “poor man’s” approach to admission control has had moderate success, it is now strained by the growth of the MBone user community and by the MBone’s transition from experimental to production service.

2.3.2 INRIA Videoconferencing System

Soon after *nv* was put to use for MBone video transmission, INRIA released their video application called the INRIA Video Conferencing System or *ivs* [167]. Unlike *nv*, *ivs* is an integrated audio/video conferencing system that relies exclusively on H.261 [171] for video compression. Because it utilizes a standardized algorithm, *ivs* interoperates with a large installed base of H.320 video codecs as an H.320-compliant bit stream is easily generated from an H.261 packet stream by introducing H.221 framing in software [77]. Also in contrast to *nv*, *ivs* makes no assumptions about the availability of network resources and adapts to available capacity through the source-based congestion control algorithm described earlier in this chapter.

Because *ivs* utilizes the temporal prediction model in H.261, its bit streams could be highly susceptible to packet loss. Since image blocks are coded differentially, a lost differential update propagates from frame to frame until an intra-mode update is received, and in H.261 this update might not occur for many tens of frames (up to 132). To counteract this problem, *ivs* uses Automatic Repeat reQuest (ARQ) on image blocks so that receivers can recover from loss. Unfortunately, as described earlier, this can result in a feedback implosion problem so *ivs* runs ARQ only for small sessions (i.e., less than ten members).

Even though the compression performance of H.261 is much better than that of the *nv* codec, *nv* was used most often by the MBone community because of its better run-time performance which, as a practical matter, translates into better end-to-end performance. More recently, H.261 has gained favor because the RTP/H.261 encapsulation protocol has been improved and a faster and more robust H.261 coder became available with our release of *vic* in 1993.

2.3.3 CU-SeeMe

Shortly after the MBone tools appeared on Unix workstations, researchers at Cornell University undertook an effort to produce similar functionality on Macintoshes. Their application, called *CU-SeeMe*, uses a simple, low-quality grayscale video coding scheme combined with a unicast-based transmission scheme over the reflector network described earlier [46]. The compression scheme downsamples the input video to 160x120 4-bit grayscale image and on each new frame conditionally replenishes every 4x4 image block that changes sufficiently.

CU-SeeMe became enormously popular because it was available on commodity hardware and did not require IP Multicast support or connectivity to the MBone, which was both administratively difficult to arrange and not universally supported on end-systems.

2.3.4 Streaming Video

While the video applications described above are all designed for real-time, interactive human-to-human communications, an equally important application is playback of pre-stored video material, e.g., for "video on demand" or for browsing video clips embedded in multimedia documents over the World Wide Web. Because the World Wide Web is based on a request/response protocol layered on top of TCP, the real-time display of video clips is easily implemented by a simple brute-force playback scheme — the client transfers the complete compressed representation of the clip in a one large request and then locally decompresses and renders the clip. However, the latency introduced by retrieving the entire stream before initiating playback can be substantial, especially for low bandwidth paths.

To reduce the start-up latency of video transfers, many Web browsers implement a "streaming playback" where the playback process is initiated before the transfer is complete. In this scheme, the receiver initiates the transfer then monitors the rate (and possibly the variance of the rate) of the received data. Assuming the transfer rate exceeds the natural playout rate of the underlying media, the playback process can start as soon as there is enough buffered data to absorb periodic stalls or slow-downs in the TCP transmission. Since the client effectively uses present and past performance observations to predict future network performance, the scheme is not robust against significant changes in network load. That is, the buffer can underflow because the actual bandwidth was less than the predicted bandwidth and an inadequate amount of data was buffered before initiating playback. Two vendors of Web-related software have recently joined forces and proposed the "Real Time Streaming Protocol" (RTSP) [140] as an open standard for controlling and configuring streaming media.

2.3.5 Digital Media Toolkits

Several research systems for networked multimedia emphasize the software architecture over the communication protocol. In these systems, a toolkit provides basic building blocks (usually objects) in the form of a code library with an application programming interface (API) to that library providing high-level abstractions for manipulating multimedia data flows. Objects are linked together through the API and distributed applications can be built by placing objects on different nodes in the network. To simplify the programming model, toolkits usually assume that communication is application-independent and offer a generic, least-common-denominator network interface built using traditional transport protocols.

The multimedia and computer-supported collaborative-work research communities have developed an array of toolkits and techniques for collaborative work. Examples include DAVE [125], SCOOT [36], GroupKit [143], the Continuous Media Toolkit (CMT) [144], and VuSystem [112]. We briefly describe CMT and VuSystem since their approaches are most similar to ours.

CMT is an object-oriented multimedia toolkit that allows objects to be mixed and matched into arbitrary applications. CMT objects are implemented in C and composed using the Tool Command Language *Tcl* [131]. Smith's Cyclic-UDP described earlier is implemented as a CMT object and more recently the CMT development team has added objects that interface with RTP and the MBone [144].

The VuSystem is a similar multimedia toolkit built on top of an object-oriented Tcl extension [112]. C++ classes implement multimedia objects that produce, consume, or filter real-time media streams and a Tcl shadow object mirrors each C++ object. Methods invoked on the Tcl shadow object are dispatched to the C++ object. Like CMT, objects can be created, composed, and configured from Tcl.

CMT, VuSystem, and our approach in the MBone tools all evolved independently of each other but converged to the same programming model: heavy-weight multimedia processing objects implemented in a compiled language (C or C++) and light-weight control functionality, user-interface abstractions, and data path glue implemented in an interpreted language (Tcl). This approach forces the programmer to factor out control and data abstractions into separate modules and thereby encourages a programming style where objects are free of built-in control policies and semantics and can thus be easily reused.

2.3.6 Summary of Internet Video Work

In this last section of our related work chapter, we discussed several well known Internet video applications that pioneered approaches for:

- large-scale, open-loop multicast transmission of video,
- source-based adaptive congestion control for video,
- combined design of the video coding algorithm with the network transmission system,
- streaming video transfers, and
- digital media toolkits.

Although these approaches demonstrate that large scale video transmission over the Internet is feasible, no existing scheme accommodates multicast transmission to receivers with heterogeneous bandwidth constraints. As we will see in later chapters, our work broadens these frontiers to encompass heterogeneity through the development of receiver-driven adaptation and layered compression.

Chapter 3

The Network Model

The work in this thesis builds heavily on the Internet Protocol architecture and the IP Multicast service model. In this chapter, we present an overview of this network model to provide context for our layered transmission protocol, RLM. Our system leverages upon the following concepts:

(1) Internet Protocol. The Internet Protocol architecture and its extension to multicast (IP Multicast) form the foundation for our communication system. Using the group-oriented communication model of IP Multicast, RLM utilizes receiver-directed group membership to *implicitly* control and fine-tune data distribution within the network.

(2) Multicast scope. In addition to receiver-directed distribution control, RLM uses source-based "multicast scope" to *explicitly* control the reach of data transmission. RLM works in concert with explicit scope mechanisms when available and administratively feasible.

(3) Routing orthogonality. In a general network, routes can be computed together with or separately from the transport protocol. We argue that route computations should be carried out orthogonal to the transport mechanisms as in the traditional Internet architecture.

(4) Lightweight Sessions and RTP. IP Multicast provides only a simple best-effort packet delivery model. We rely on additional application structure, called the "Lightweight Sessions" (LWS) architecture, to provide end-to-end delivery of real-time interactive media. LWS builds on both IP Multicast and the Real-time Transport Protocol (RTP).

(5) Multiple Multicast Groups. RLM requires that layered streams be selectively forwarded at strategic locations within the network. Consequently, individual layers must be distinguished at the network layer. We therefore use distinct multicast groups, rather than a traditional transport demultiplexing key, to distinguish each layer of the distribution hierarchy.

(6) Best-effort Packet scheduling. The end-to-end performance of a transport protocol depends on the algorithms used by routers to schedule packets for transmission within the network. We survey the common approaches to packet scheduling and argue that drop-priority scheduling provokes undesirable end-system behavior and consequently should not be deployed within a best effort service class.

In the following sections, we describe each of these architectural components that together form the foundation for our layered transmission system.

3.1 The Internet Protocol

The Internet Protocol (IP) architecture defines the mechanisms and protocols for delivering messages between end-systems in an interconnected "network of networks". Each communicating end-system or *host* has one or more interfaces that attach it to one or more networks and each interface is identified by a globally unique *address*. The address of the interface consists of a hierarchically structured network number that identifies a particular network within the internetwork and a host number that identifies a particular host within the network.

The fundamental message delivery unit in IP is called a *datagram*, while the fundamental transmission unit is called a *packet*. Datagrams are usually mapped directly onto packets, but if a particular datagram is larger than the maximum transmission unit of the underlying network, then the datagram is *fragmented* into multiple packets. The destination host reassembles the fragmented datagram before delivering it across the IP service interface to the application.

An IP *router* has two or more interfaces that attach it to multiple networks. Because of these multiple attachments, a router can *switch* or *forward* packets between networks. All of the routers in the network cooperate to deliver packets from an arbitrary sender to an arbitrary destination. Each router maintains a table of *routes* that maps a destination network to an outgoing interface. When a packet arrives on an incoming link, the router locates the proper route and forwards the packet toward its destination on the corresponding interface. Routers exchange control messages to effect a distributed algorithm that causes each router to learn its local representation of a global set of paths through the network. Typically, this *routing protocol* causes the aggregate set of routing tables to converge to a state where any packet will traverse the "shortest path" between any two hosts attached to the network.

The traditional Internet service model is *best-effort*. The network does not guarantee that packets necessarily reach their destination (i.e., they can be dropped because of errors or buffer overflow) nor does it guarantee that packets are delivered in the order they were produced. Rather, routers are engineered to make a reasonable, best-effort attempt at delivering packets toward their destination. Because these delivery semantics place few constraints on the design of the network layer, extremely robust systems can be built on this model. For example, if a link fails, the routing protocol inevitably takes time to compute new routes around the failure. During this time, packets may be reordered, lost in routing loops, or even duplicated. Because the service model does not preclude this behavior, an application or transport protocol built on top of IP must be made robust to these events and, as a result, failures are gracefully tolerated.

In contrast, networks that guarantee in-order, non-duplicate packet delivery are burdened with complex error-recovery schemes. When a link fails, the protocol must re-route the traffic without reordering packets. For example, ATM networks are connection-oriented and guarantee in-order delivery within a connection. Rather than implement error-recovery, ATM implementations typically reset all connections that flow through a link that fails.

3.2 IP Multicast

The architecture described above handles *unicast* transmission, where a single source transmits datagrams to a single destination. But some applications require *multicast* delivery, where a source transmits packets simultaneously to an arbitrary number of receivers. Clearly one can implement multicast out of unicast by sending a copy of each datagram to every receiver, but this is extremely inefficient because multiple copies of each packet will traverse the same underlying links. Instead, a much more efficient technique is to replicate packets only at fan-out points in the network so that at most one copy of each packet appears on a given link. Deering proposed this approach in the *IP Multicast* service model, which extends the traditional IP unicast delivery semantics for efficient multipoint transmission [41]. In IP Multicast, packets are forwarded along a *distribution tree*, rooted at the source of the data and extending to each receiver in the multicast group.

A key feature of IP Multicast is the level of indirection provided by its *host group* abstraction. In this *group-oriented* communication framework, senders need not know explicitly about receivers and receivers need not know about senders. Instead, a sender simply transmits packets to an IP *group address* and receivers tell the network (via the Internet Group Management Protocol or IGMP [52]) that they are interested in receiving packets sent to that group. Moreover, the process by which receivers join and leave multicast groups is timely and efficient (on the order of a few milliseconds).

A number of multicast routing protocols compute spanning trees from an anchor point in the network (either the sending host or a "rendezvous point" or "core") to all of the receivers, e.g., Protocol Independent Multicast (PIM) [40], Distance Vector Multicast Routing Protocol (DVMRP) [174], or Core Based Trees (CBT) [7]. Because the anchor point (i.e., source address) uniquely identifies the spanning tree, multicast routers can use it to compute the forwarding decision for a given packet. That is, to forward a packet, a multicast router indexes a routing table using the packet's source address, which yields a routing entry containing a set of outgoing links. The router then transmits a copy of the packet along each outgoing link. To avoid sending traffic where there are no interested receivers, a multicast router omits links that have no downstream receivers for a particular group (e.g., the router might keep a list of pruned groups for each outbound link). In short, the *source* address determines the routing decision and the *destination* address determines the prune decision. This contrasts with the unicast case where routing is (generally) determined only by the destination address.

3.2.1 Group Maintenance

As we will see in later chapters, the performance of RLM depends critically on the multicast join/leave latencies of the group maintenance protocol. Once a receiver leaves a group, the network must suppress the flow in a timely manner because congestion persists as long as the network continues to deliver the offending layer. Similarly, to allow receivers to correlate their actions with resulting congestion periods, the network must instantiate a new flow expediently. Fortunately, IGMP carries out both of these operations on reasonable time scales. When a receiver joins a new group, the host immediately informs the next-hop router, which in turn, immediately propagates a *graft* message up the multicast distribution tree in order to instantiate the new group. If the flow already exists, the graft is suppressed. When a receiver leaves a group, the situation is more complex because the next-hop router must determine when all the hosts on a subnet have left that group. To do this, when a host drops a group, it broadcasts a "leave group" message on the subnet and the router responds by briefly accelerating its normal membership query algorithm. Upon quickly determining that no members remain, the router sends a *prune* message up the distribution tree to suppress the group.

Figure 3.1 illustrates how the group membership protocol evolves over time for a simple scenario. A source S transmits packets to some group. Initially, hosts R_1 and R_2 issue group membership requests to the network using IGMP. Each last hop router sees the IGMP message and propagates a graft message up the distribution tree toward S. In the time it takes to send the message up the tree, the paths are grafted on and data begins to flow down the tree as illustrated in the left half of the diagram.

At some later point in time, R_1 and R_2 leave the group again using IGMP. After a timely exchange of messages (usually no more than a few milliseconds), each leaf router determines that there are no downstream hosts subscribed to the group. In turn, they prune the group by not forwarding packets sent to that

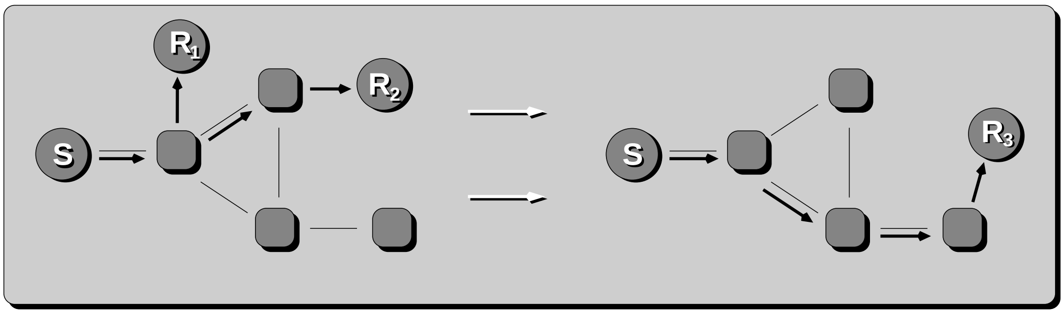


Figure 3.1: **Dynamic Group Membership.** IP Multicast provides a communication model where sources and receivers need not explicitly know about each other. Instead, hosts join a multicast group and the network delivers packets sent to that group to all interested receivers. The membership protocol is dynamic and timely. For example, if receivers R_1 and R_2 leave the group while receiver R_3 joins, the network will prune back the distribution path from R_1 and R_2 and expediently graft on the flow to R_3.

particular group on the pruned interface. Moreover, if a router receives a multicast packet addressed to a group that is pruned from all outgoing interfaces, it propagates a prune message further up the distribution tree. In this data-driven fashion, traffic is pruned back to the earliest point in the spanning tree.

At the same time the flows to R_1 and R_2 are pruned back, a third receiver R_3 may decide to receive traffic sent the group. Again, it uses IGMP to join the group and the new path is grafted on, leading to the configuration depicted in the right half of Figure 3.1.

3.2.2 Multicast Scope

While receiver interest *implicitly* constrains a multicast flow through the network's use of traffic pruning, multicast scope *explicitly* limits a stream to a topologically constrained set of receivers. For example, a university lecture that is of interest only to a local campus community should be broadcast only to that site rather than entire world. A source may exploit two classes of scope control to constrain its traffic:

- distance-based scope, and
- administrative scope.

In distance-based scope, the time-to-live (TTL) field in the IP packet header constrains the distribution of a packet. The TTL limits the number of forwarding hops that a packet can sustain before being dropped. The source host sets the TTL field to some initial value and each router along the path decrements the TTL by one. If the TTL ever reaches zero, the packet is discarded. This mechanism was originally introduced to prevent packets from circulating forever in a routing loop[1].

Distance-based scopes are implemented by assigning a *threshold* to each multicast link. If a multicast packet's TTL is less than the threshold, then the packet is dropped. By judiciously choosing the thresholds, hierarchical scope regions can be defined. In the MBone, the accepted convention establishes three

[1] The TTL is also decremented at least once per second to limit the amount of time a packet can exist within the network, ensuring to higher-level protocols that a duplicate packet will arrive within a bounded time window of the original.

universal scopes: the "local site" (threshold 16), the "local region" (64), and the "world" (128). A fourth, less often used scope is the "low-bandwidth world" (> 128). Each site configures its multicast topology so that all external links have a threshold of 16, thereby constraining all packets with TTL less than 16 to the local site. Because regional and worldwide traffic originates with a TTL greater than 16, it is forwarded across the local site's boundary.

Since TTL threshold checks are based on simple numerical inequalities, a distance-scope can only ever become more restricted as a packet travels farther from the source. Hence, distance-based scope regions can only be arranged in a containment hierarchy, where the intersection of any two scope regions must be exactly the smaller region. For example, an individual subnet cannot simultaneously be in an "LBL-scope" and a "UCB-scope" unless the LBL-scope is contained entirely within the UCB-scope, or vice versa.

Deering proposed an administrative scoping scheme that provides overlapped regions by associating the scope region with a multicast address rather than a TTL [41, §3.2], and Jacobson and Deering presented specific mechanisms for incrementally realizing this scheme [89]. Here administrative scopes are imposed by configuring administrative boundaries at the borders between organizations. A special block of multicast addresses is reserved for administrative scope and since administratively scoped traffic does not flow across boundaries, scoped addresses need not be unique across organizational boundaries.

Unlike distance-based scopes that simply shed traffic at scope boundaries, administrative scope actively complements the normal multicast routing machinery. At an administrative boundary, unwanted traffic is pruned back up the distribution tree to prevent packets from unnecessarily flowing all the way to the edge of the boundary. In contrast, a router at the edge of a distance-based scope boundary cannot prune a flow because the TTL field in the underlying packet stream can vary dynamically and its value is a function of the source not of the destination group.

Administrative and distance-based scope mechanisms are complementary to our layered compression model. Although our receiver-driven scheme principally uses group membership to limit distribution, the approach can be augmented by scope controls when we have explicit knowledge of topological bandwidth constraints. If we have a priori knowledge of the underlying topology, we can configure our transport protocol to avoid unnecessary adaptation in certain cases. For example, in the UCB seminar transmissions, we could use administrative scopes to limit the high bandwidth streams to the local campus. Since we know access bandwidth to the Internet is limited, receivers need not adapt beyond this level — scope control can explicitly convey this constraint.

3.3 Orthogonality of Routing

Because our network adaptation scheme depends on the particular path a traffic flow takes through the network, the system is sensitive to the network routing infrastructure. However, we view the route computation problem as "orthogonal" to the end-to-end adaptation algorithm. The IP service model very deliberately imposes a separation between the packet-routing and packet-forwarding functions and rather than jointly optimize the route computation with application requirements, we take the set of multicast routes computed by the network as given.

Because the IP service model says nothing about how groups are routed through the network, multicast routing protocols are free to route different multicast groups along different distribution trees. In fact, aggregate network performance can potentially be improved with *policy-based routing*, where routes may depend on the underlying traffic type. For example, different multicast groups might be routed along different trees for load balancing. Therefore, we cannot guarantee that all the groups of a single session follow the same distribution tree. While our network protocols are most easily reasoned about when all the groups follow the same paths, this is not a requirement as we will later see.

By separating the route computation from our end-to-end adaptation scheme, we preclude optimizations that adjust routes to achieve better performance. Although integrating routing decisions with our end-to-end adaptation appears attractive on the surface, we believe that the benefits accrued through their separation outweigh those of a combined design for a number of reasons:

- The underlying network topology might not be fixed or packets might be striped across multiple paths.
- Routing state is typically distributed and hierarchical, i.e., routes internal to an administrative domain are not advertised across the boundary and are instead aggregated into a single advertisement. Hence communicating detailed topology information to the end host is not always possible.
- In a global internetwork, routing information is voluminous and dynamic. It would be difficult if not impossible to maintain this information at each end-system in the network. Thus a scheme that relies on global network topology to carry out optimizations is inherently non-scalable.
- Because the current IP Multicast service model does not export its underlying routing mechanisms, the end-host has no control over how multicast groups are routed through the network.

Together, these points support the classic "end-to-end argument" [147]. Any functionality that can be implemented efficiently at higher layers, outside of the network, should not be built into the network. To this end, one of our principal goals is to produce a system that works well in the current environment, with all new functionality built on top of the existing network. If necessary, we might later employ optimizations within the network to improve performance, e.g., policy-based routing techniques that explicitly account for the interaction between our application and the network. The *tentative-join* machinery described in the next chapter is one such approach.

3.4 Lightweight Sessions

Because IP Multicast provides only a simple "send a packet" interface, richer application semantics must be built from this simple primitive. One step toward a more structured architecture is the "Lightweight Sessions" (LWS) model that underlies the MBone tools and builds heavily on IP Multicast. In this framework, a *session* is a collection of end-hosts that communicate using a particular, somehow agreed upon, set of IP Multicast group addresses. The session rendezvous problem — i.e., the task of each participant locating all others in a distributed group — is greatly simplified by the level of indirection provided by multicast groups. Here each participant must merely learn the addresses that correspond to the session using a simple Announce/Listen protocol [148] running on a well-known multicast group. The Session Description Protocol (SDP) [75] is a proposed standard that defines the message formats and protocol for advertising the multicast address bindings.

In LWS, group management is not explicit. Unlike many traditional conference management schemes, there is no protocol for controlling access to the group or for maintaining a consistent group view at each receiver. Privacy is achieved not through access control, but instead through end-to-end encryption [91]. Similarly, conference control mechanisms are effected through the "receiver orchestration" techniques discussed in Chapter 7. This style of conference management is often characterized as *loosely-coupled* control because explicit synchronization among group members is unnecessary.

3.4.1 The Real-time Transport Protocol

While LWS provides a framework for group communication, it does not specify how the members in the group communicate with each other. Within LWS and on top of the IP service interface we must define specific transport protocols and data formats to facilitate communication, and we must further determine a method for mapping these data formats onto multicast transmission channels.

Through our work on the MBone tools and other similar efforts [164, 61, 149, 150], the following communication model emerged. A session is comprised of a number of media and each media type is allocated to two transport channels — one for data and one for control — each using the same multicast group. Unlike traditional audio/video conferencing systems like H.320 and MPEG that multiplex different media by interleaving their bit streams, LWS streams are independently transmitted and multiplexed only in the sense that the underlying packet streams share the same physical network. Traditional transport protocols are typically implemented within the operating system kernel and entail "hard" connection state, whereas the LWS model for transport is relatively "thin" and is implemented within the application. This approach is embodied in the Real-time Transport Protocol or RTP [153], recently standardized by the Audio/Video Transport Working group of the Internet Engineering Task Force (IETF).

RTP defines much of the protocol architecture necessary for video transmission over a multi-point packet network. An "RTP session" is a component of an "LWS session" and represents a collection of two or more end-systems exchanging a single media type and related control information over two distinct underlying transport channels. For UDP [136] over IP Multicast, these two underlying transport channels are mapped onto two distinct UDP port numbers sharing a common multicast address. An active source transmits its signal by generating packets on the data channel that conform to the payload format specification for the underlying compression format. Simultaneously, all of the end-systems in a session exchange information over the control channel. Periodically, each source generates a Real-time Transport Control Protocol or RTCP message. These messages provide mechanisms for sender identification, data distribution monitoring and debugging, cross-media synchronization, and so forth.

Each source in a session is identified by a 32-bit Source-ID. Source-ID's are allocated randomly and conflicts are handled by a resolution algorithm. Random allocation avoids complex global synchronization and consistency algorithms. Since conflicts can cause Source-ID's to change at any time, a source's "canonical name" or CNAME provides a persistent and globally unique identifier. Data packets are identified only by their Source-ID but the RTCP control messages include the binding between CNAME and Source-ID. The CNAME is a variable length ASCII string.

Data packets also contain a media specific time stamp (e.g., a sample counter for audio and a frame clock for video). RTCP packets advertise the mapping between media time and the sender's real-time clock. To counteract delay variances induced by the network, each receiver dynamically adjusts the amount of playback buffering in order to recover the sender's original timing while minimizing delay. This "playback point algorithm" can be extended to carry out cross-media synchronization by aligning each individual media with the media that has the maximal playback point [88].

Since RTP is independent of the underlying network technology, it simultaneously supports multiple network protocols. Figure 3.2 illustrates how RTP fits into several protocol stacks. For IP and IP Multicast, RTP is layered over UDP, while in the Tenet protocols, it runs over RMTP/RTIP [8]. Similarly, applications can run directly over an ATM Adaptation Layer. In all these cases, RTP is realized in the application itself.

RTP		
UDP	RMTP	AAL5
IP	RTIP	ATM

Figure 3.2: **RTP and the Protocol Stack.** RTP is independent of the underlying transport protocol. This feature of RTP has allowed our tool *vic* to be successfully deployed over the Tenet Real-time Message Transport Protocol (RMTP) [8] and the ATM Adaptation Layer (AAL5) in addition to its commonly used configuration of UDP over IP Multicast.

3.5 Multiple Multicast Groups

A network host typically differentiates multiple incoming data flows with a transport-level demultiplexing identifier usually called a *port*. Accordingly, one approach for transmitting layered streams over multiple network channels is to use a range of transport ports, where each port identifies a separate layer. However, our network protocol relies on the network's ability to selectively forward layers at particular points inside the network. Since the decision to forward a packet along a given link of the multicast distribution tree is fundamentally a routing decision, the different layers must be explicitly visible to the routing or network layer. We therefore use distinct multicast addresses — i.e., network-visible entities — to identify each subflow.

As discussed in later chapters, the strict end-to-end approach of RLM could be augmented with new mechanisms in the network. In this case, the network may need to infer structure in the set of multicast groups that make up a layered transmission, e.g., to allow the network to perform a control action on the base layer. Rather than devise a specific protocol between the end-systems and network to convey this structure, we propose that the structure be reflected explicitly in the address. That is, we can reserve a block of multicast addresses for layered stream transmission and adopt the convention that the least significant N bits would identify the layer number. For example, the base layer (i.e., layer 0) could be determined from any of the enhancement layer addresses by setting the N low bits to 0. While the current IP version 4 multicast address space is relatively constrained, addresses in the forthcoming version 6 of IP are 128 bits wide and have adequate range to impose such a structure.

Since we use multiple multicast addresses for layered distribution, we need a mechanism for allocating and distributing the addresses for a given session. Currently, SDP provides such a mechanism, but is restricted to contiguous ranges of addresses. This is problematic if we want to apportion some layers of the distribution to administratively-scoped addresses and others to distance-scoped addresses. Moreover, the allocation of administratively-scoped addresses within each region is local (since scoped addresses do not cross boundaries), which implies that one global SDP advertisement for the layered session is inadequate. Clearly, SDP must be extended with a capability for describing more flexible layered stream structures. At present, this is an open design issue.

3.6 Packet Scheduling

Several research efforts on integrated services networks introduce new mechanism in the network to give preferential treatment to real-time traffic. A key focus here is on the algorithms used to *schedule* packets at the routers within the network. In this section, we briefly survey several approaches to packet scheduling and, in particular, describe flaws in and warn against the use of mechanisms that differentiate traffic types in a best-effort environment.

3.6.1 Random Drop

One of the simplest scheduling algorithms is to serve packets in first-in first-out (FIFO) order. Because a router's buffer memory is finite and because arbitrarily long queues cause undesirable delays, the queue length must be restricted and consequently packets must be dropped when the queue becomes full. The simplest approach for selecting a packet to drop — called *drop-tail* — discards the packet from the end of the queue (i.e., the packet that just arrived and caused the queue overflow).

Because packets are dropped deterministically from the end of the queue, end-system protocols can interact adversely with the network and suboptimal traffic patterns can emerge. For example, Floyd and Jacobson found that TCP connections can suffer pathological phase synchronization effects that bias the throughput of some connections over others [55]. They and others have proposed that the impact of these adverse effects can be reduced by replacing the drop-tail policy with *random-drop*, where the dropped packet is chosen at random from the queue [115].

Under both drop-tail and random-drop, packets are dropped at a congested router in an uncontrolled fashion. Hence, for layered packet transmission, delivered quality will rapidly degrade under congestion because packets are dropped uniformly across all layers.

3.6.2 Priority Drop

Instead of the best-effort model described above, the universally cited approach to layered packet transmission adds a drop-preference packet discard policy to all the routers in the network. Under drop-preference, when congestion occurs, routers discard less important information (i.e., low-priority packets) before more important information (i.e., high-priority packets). For example, a layered video stream is carried by a drop-priority network by placing highest priority on the base layer and lower priorities on the enhancement layers. Consequently, under congestion, the more important base layer survives unscathed while the enhancement packets are dropped. Although this approach provides graceful degradation in the presence of packet loss, we believe it has scaling problems because it rewards poorly-behaved users.

This effect is illustrated in Figure 3.3, which plots the quality of a received signal versus the requested bit rate for both priority-drop and random-drop policies. In both cases, the quality of the received signal increases with the requested rate up to the bottleneck capacity B but beyond this, the quality depends on the drop policy. With random-drop, quality degrades because packets are dropped uniformly across all layers, while with priority-drop, quality remains constant because only "enhancement" packets are dropped. The key distinguishing feature of these two curves is their convexity. Because the random-drop curve is strictly convex, it has a unique maximum. Thus we can design a control system that maximizes the quality metric and drives the system toward the stable, uncongested bottleneck rate B. The priority-drop curve has no unique maximum and hence does not admit a control system that optimizes delivered quality by converging to a single, stable operating point. In fact, a greedy or naive user would likely request a rate far above the bottleneck rate B, driving the network into a persistently congested state.

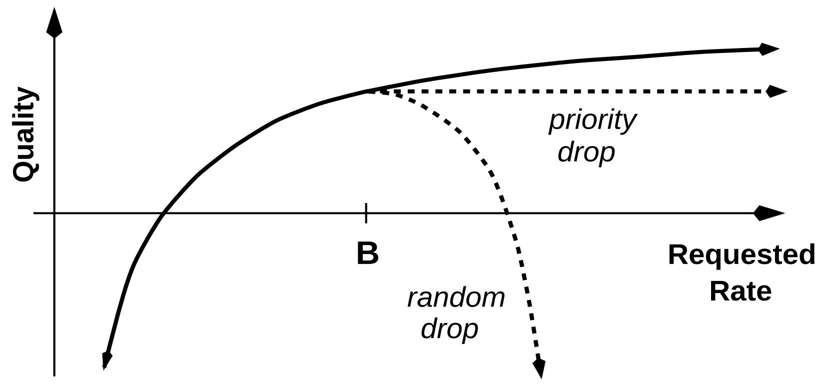


Figure 3.3: **Priority-/Random-drop Tradeoff.** These curves qualitatively illustrate the negative consequence of drop priorities in best-effort networks. The upper curve represents the information theoretic rate-distortion curve of a communication signal like video. As we increase the rate of transmission, the signal quality improves with diminishing returns. In a real network, however, the quality improves only up to the bottleneck capacity B. Beyond this, quality either degrades rapidly (under random-drop) or stays constant (under priority-drop). Unfortunately, under priority-drop end-users have no incentive to operate at or below B and will more likely operate above B, leading to persistent congestion.

To summarize, we believe that drop priorities should not be incorporated into best-effort networks as has been preliminarily proposed in Version 6 of the Internet Protocol [42, § 7]. Instead, the best-effort service class should provide uniform service to all traffic (within the class) and enhanced services should optionally be provided through separate service interfaces, e.g., via RSVP.

3.6.3 Random Early Detection

Although random-drop gives advantages over the drop-tail packet discard policy, we can improve performance further with a variant of random-drop called Random Early Detection or "RED" [56]. Under both drop-tail and random-drop, routers drop packets only after their queue is full. This widely deployed drop-tail discard policy is unfortunate because it delays the "congestion warning signal" from the receivers until well after congestion has occurred. RED gateways, on the other hand, react to incipient congestion by discarding packets at the onset of congestion (i.e., when the average queue size exceeds some threshold set much lower than maximum queue size). RED's early reaction to congestion interacts nicely with our layered transmission protocol because it allows receivers to react to congestion before the bottleneck link becomes fully saturated.

3.6.4 Fairness

In a network with arbitrary numbers of senders each transmitting to an arbitrary number of receivers, each receiver should individually adjust its number of layers so that the aggregate system performance is "good". When there is only a single source sending to some number of receivers, "good" is well-defined: each receiver should receive the maximum number of layers that the network can deliver. But when there are multiple sources, "good" is ill-defined because it depends on the relative importance of the users

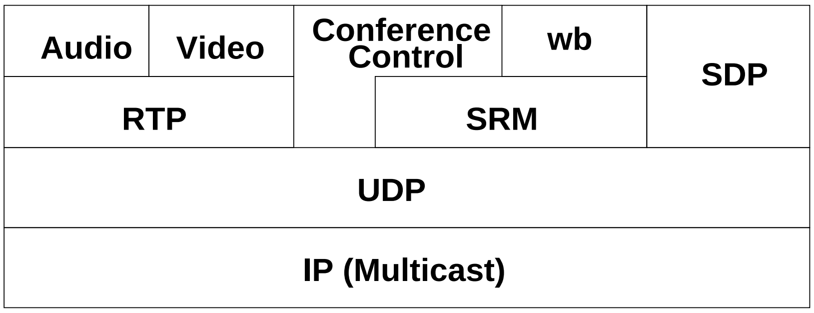


Figure 3.4: **LWS Conferencing Architecture.** The Lightweight Sessions Conferencing architecture leverages off the Internet Protocol architecture and the IP Multicast service model.

within and across sessions. In short, an aggregate performance metric depends on how group "fairness" is defined.

Rather than tackle the problem of defining fairness, we place our initial focus on the design of layered transmission protocol in *isolation*, that is, when a single source sends to multiple receivers without interfering traffic. Once we have a stable and scalable solution for this simple case, we can address the dynamics that result from interactions among multiple instances of our protocol or between our protocol and other protocols like TCP. Our goal is to design the system to interact gracefully with competitive traffic by reacting conservatively enough to congestion so that the aggregate system is "well behaved" and stable. Unfortunately, we can say little about the stable operating point that our protocol by itself would obtain. In general it is not possible to achieve a "fair" allocation of bandwidth without some additional machinery in the network, even if all the end-nodes cooperate [92]. But, if machinery for fairness is added to the network, our protocol could work effectively in concert with it.

Similar circumstance surrounds the design of TCP. TCP congestion control works well in isolation but in aggregation can be unfair [55]. As an optimization, network mechanisms can be introduced to make TCP perform better: RED gateways or Fair Queuing (FQ) [45] routers minimize the interaction between connections to improve aggregate system performance. Similarly, we can design our protocol to behave relatively well in a loosely controlled, drop-tail, best-effort network, and as an optimization add RED or FQ to the network (or to parts of the network) to improve aggregate performance.

3.7 Summary

To close this chapter, we illustrate the LWS multimedia communications architecture in Figure 3.4 using a traditional "protocol stack" diagram. The bedrock of LWS is the Internet Protocol architecture and its extension to multicast. In between IP and the application-level protocols sits a thin UDP layer to provide simple process level demultiplexing at the end-systems. The Real-time Transport Protocol provides a layer above UDP for transmission of real-time audio/video streams. Similarly, shared whiteboard (wb) data is distributed using the Scalable Reliable Multicast (SRM) mechanisms mentioned in Chapter 1, while "conference control" protocols, running directly over UDP or on top of SRM, glue all the components together within the end-systems and across the network. Finally, sessions are advertised using the session description protocol running on a well-known IP Multicast address. In short, these components fill in a multitude of protocol details requisite to an effective and scalable multimedia communication systems, and altogether comprise a solid foundation for our layered transmission architecture.

Chapter 4

Joint Design across Network and Application

Our layered compression/transmission architecture for Internet video is an example of a combined design that explicitly accounts for interaction among its constituent components. We do not focus on the video compression design in isolation or on a high-performance transport protocol without considering the semantics of the applications that it carries. By accounting for the interaction among sub-components, we produce an overall system with high performance not only over the network but also through the end-system and ultimately to the user. This approach of intermodular optimization is a general concept that covers systems-oriented design techniques across many fields. In particular, two analogous design methodologies have emerged independently in two very related yet segregated fields: *joint source/channel coding* (JSCC) from traditional communications theory and *application level framing* (ALF) from computer network design. This chapter develops the idea that ALF and JSCC are parallel concepts.

The notion that an application's semantics should be reflected in the design of its network protocol is embodied in Clark and Tennenhouse's ALF protocol architecture [31]. Heybey's thesis explores the integration of video coding algorithms into the ALF framework by adopting existing compression standards (e.g., JPEG, H.261, and MPEG) without modification [81]. In contrast, we elaborate ALF with the premise that we should not only optimize the network protocol for the application, but we should also optimize the application for the network. Heybey's approach treats the compression algorithm as an immutable black box. Rather than adapt the source-coding algorithm to the network, he develops intelligent framing techniques for the fixed bit syntax produced by the black box. Conversely, we propose to break open the box and tailor its behavior for the network through modification or restriction of the underlying compression algorithm.

While the ALF design approach has just recently gained momentum in the network research community, an independent manifestation of the same basic concept appeared long ago in the context of communications theory. Joint source/channel coding states that one can often achieve better system performance by combining the design of compression and error-control coding rather than treating these sub-problems independently. Garrett and Vetterli [63, 64] first applied this terminology to packet networks by viewing the packet transport mechanism as the channel-coding algorithm. In his thesis, Garrett argues that real-time traffic is better served by jointly designing the compression algorithm with the transmission system [66].

We believe that the emergence of JSCC and ALF signals a shift in the core nature of network research — the traditional, highly layered protocol architecture must be abandoned. Rather than design protocols composed of modular black boxes, we must synthesize new, application-specific protocols (e.g., FLIIT [37], RLM, RTP [153], and SRM [59]) that account for the interaction between application and network.

In this chapter, we establish ALF and JSCC as a core design principle for our layered video architecture. To this end, we present our earlier work on the non-layered *Intra-H.261* format that forged our approach to ALF/JSCC-based design. Not only did the design decisions behind Intra-H.261 influence the layered codec described in Chapter 6, but Intra-H.261 has proven successful in its own right and is in widespread use for Internet video transmission. The contributions described in this chapter include:

- our development of a synthetic view of JSCC and ALF;
- our extension to the ALF protocol architecture, which says that we not only should design the network protocol to reflect application semantics but we should also design the application to explicitly account for the limitations and difficulty of network communication;
- our quantification of macroblock-based framing and conditional replenishment as specific instances of the JSCC/ALF approach that improve end-to-end performance over the Internet; and
- our synthesis of ALF/JSCC principles in the adaptation of the ITU H.261 video compression standard for Internet video transmission. Our scheme called "Intra-H.261" led to improvements in the RTP Payload Specification for H.261 [166].

4.1 The Separation Principle

Before we can precisely define joint source/channel coding, we must first describe Shannon's *separation principle*. In Shannon's framework, an information source produces a string of symbols from some fixed alphabet and the *source-coding* problem is to encode the source symbols in a series of codewords that minimizes the average codeword length (based on the given probability distribution of input symbols), i.e., source-coding *is* data compression. In turn, the source symbols are transmitted across a noisy communication channel that corrupts symbols according to some probabilistic model. Even in the face of such indeterminacy, Shannon showed that the probability of receiving a symbol incorrectly could be made arbitrarily small by introducing a *channel-coding* stage. In channel-coding, the source codewords are mapped onto channel codes that provide error-correcting capabilities. By appending redundant bits to each codeword or to collections of codewords, we can detect and potentially correct transmission bit errors. While the ability to transmit an information source over a noisy channel with negligible probability of error was a remarkable achievement for its time, even more remarkably, Shannon showed that the two problems of compression and error-correction could be separated without sacrificing optimality.

This separation principle is illustrated in Figure 4.1. Separation says that under certain conditions we can separate the overall information transmission problem into a source-coding problem and a channel-coding problem, optimize these sub-problems independently, and combine their results into an overall optimal system. In other words, a scheme that uses source-coding followed by channel-coding can be made as efficient as any single-stage source-/channel-coding algorithm.

This seminal work has profoundly influenced communications research as well as its practice. Not only is there a divide between the source-coding and channel-coding research communities, but some communication standards explicitly reflect this split. For example, the ITU H.320 standard for audio/video teleconferencing over ISDN embodies the separation principle, as H.320 is comprised of a source-coding specification for video (H.261) and an independent channel-coding specification for demultiplexing and error-control coding (H.221).

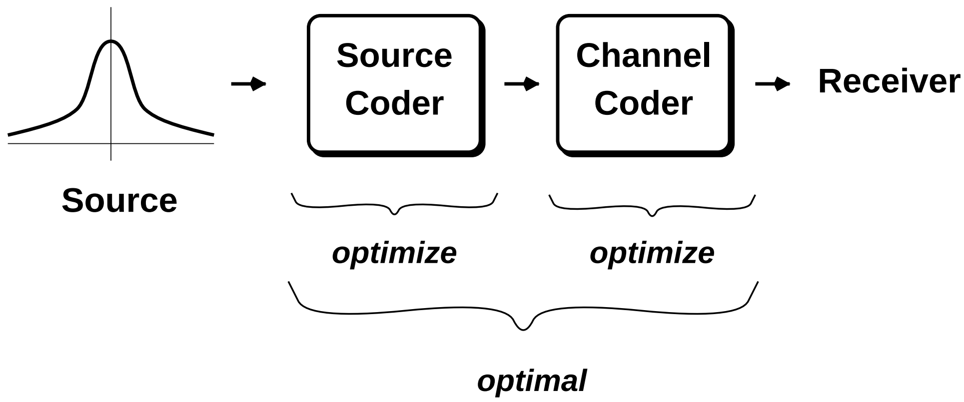


Figure 4.1: **The Separation Principle.** Shannon's *separation principle* says that under certain source and channel models we can optimize the source coding algorithm (i.e., the compression algorithm) and the channel coding algorithm (i.e., the transmission algorithm/protocol) independently. If we cascade these two modules, the overall system is optimal.

4.2 Joint Source/Channel Coding

Because the separation principle is based on arguments that hold only in the limit, a channel-coding scheme converges to the theoretic optimum only if we process an arbitrarily large amount of data and are willing to incur unbounded coding delays. In other words, separation assumes that we do not impose a delay constraint at the source. For H.320, large encoding delays are avoidable because transmission errors in the underlying ISDN circuit occur at a bit-level granularity (not typically in bursts) and can be corrected by simple and short block error codes.

On the other hand, separation does not necessarily apply to other typical communication environments. Cover showed that, for a certain model of broadcast channel, performance can be improved by superimposing the delivery of low-rate and high-rate information [35], e.g., by distributing video to heterogeneous receivers in a layered format. Hence, performance can be improved by modifying the source-coding algorithm (i.e., layered compression) to better match the channel-coding algorithm (i.e., layered transmission). In other words, we achieve improved performance through joint source/channel coding, and clearly, our proposed layered video architecture is an instance of JSCC.

4.3 Application Level Framing

While JSCC was established by theoretical work in the communications research community, the parallel notion of application level framing arose from engineering principles developed in the data networking community. To illustrate the reasoning that leads us to ALF-based design, we propose an anecdotal design of an Internet unicast video delivery system. Under the premise of separation, such a system might be designed using TCP for the channel-coding algorithm and H.261 for the source-coding algorithm. Although each of these schemes has been independently optimized and performs extremely well in its targeted environment, they interact poorly together.

TCP is enormously successful as the transport protocol that has supported the Internet through the large traffic loads currently caused by the World Wide Web. Because Internet transmission errors are manifested as burst erasures (i.e., packet losses), TCP would have to impose many packet delays at the source to compute effective error-correcting codes at the packet granularity. Moreover, the efficiency of such a code depends strongly on the accuracy of the channel model known to the source (e.g., when packet loss rates are high, the data requires more protection, while when losses are low, the data requires little or no protection). Because residual capacity in the Internet is highly dynamic, it is difficult to estimate the channel model at the source. For these reasons, TCP uses Automatic Repeat/reQuest (ARQ) instead of error-correcting codes to combat transmission errors and network buffer overflows, and includes a set of congestion control mechanisms that interact tightly with its ARQ algorithms. Together these algorithms comprise a transport protocol — the analog of a channel-coding algorithm — that is highly optimized for packet-switched best-effort networks like the Internet.

Even though H.261 and TCP each perform well in their respective environments, their combination for Internet video transmission leads to poor end-to-end performance for the following reasons:

- The ARQ mechanisms in TCP use acknowledgments and retransmissions to guarantee delivery of data. Packet retransmissions cause large variation in packet delivery times, which in turn cause stalls in the ultimate presentation of the video.

- Interactive video streams require timely delivery where it is better to discard late data rather than to wait for a retransmission. In effect, video is loss-tolerant because we can simply suffer a momentary degradation in quality rather than wait for a retransmitted packet. Since TCP provides just one specific delivery semantics (i.e., a reliable byte stream), an application cannot instruct the transport layer to cancel a retransmission for a given piece of data. Instead, it must wait for the retransmission to complete before receiving subsequent data.

- When packets are lost or reordered in the network, they arrive at the receiver in an arbitrary order. But often performance is improved by processing data as soon as it arrives and then patching any errors after the fact when the retransmissions arrive. Because TCP provides only in-order delivery, packets are never delivered out of order and thus the application cannot immediately consume misordered data even when it is timely and useful.

- TCP provides a byte stream-oriented API that hides the underlying packet representation from the application. If the application could control how data was mapped onto packets, it could use intelligent framing to minimize the impact of loss by allowing the H.261 decoder to proceed in the presence of missing packets.

Fortunately, we can solve all of these problems with application level framing. ALF leads to a design where the application takes an active role in the encapsulation of its data into network packets, and hence, can optimize for loss recovery through intelligent fragmentation and framing. In this model, the data is tailored for the network by explicitly defining the Application Data Unit, or ADU, into the application/network interface. In the next section, we describe how the ALF model explicitly accounts for interaction between the application and network to improve the end-to-end performance of packet video.

4.4 Integrated Design with ALF/JSCC: The Real-time Transport Protocol

About the same time that Clark and Tennenhouse proposed ALF, we and others developed a number of tools to explore the problem of interactive audio and video transport across packet-switched networks [90, 149, 61, 117, 164, 150]. After several design iterations over transport protocols for several different audio/video compression formats, it became clear that a "one size fits all" protocol was inadequate [59, 120]. Instead, a framework based on ALF emerged where a "thin" base protocol defines the core mechanisms and profile extensions define application-specific semantics. As described in the previous chapter, the Audio/Video Transport Working Group of the Internet Engineering Task Force (IETF) standardized this base protocol in the "Real-time Transport Protocol" or RTP [153] and developed a profile for Audio and Video conferences with minimal control [152] along with a number of payload format standards for specific applications like H.261, JPEG, and MPEG.

A key goal in RTP is to provide a very thin transport layer without overly restricting the application designer. The protocol specification itself states that "RTP is intended to be malleable to provide the information required by a particular application and will often be integrated into the application processing rather than being implemented as a separate layer." In accordance with ALF, the definitions of several fields in the RTP header are deferred to an "RTP Profile" document, which specifies their semantics according to the given application. For example, the RTP header contains a generic "marker" bit that in an audio packet indicates the start of a talk spurt but in a video packet indicates the end of a frame. The interpretation of fields may be further refined by the "Payload Format Specification". For example, an audio payload might define the RTP timestamp as a audio sample counter while the MPEG/RTP specification [82] defines it as the "Presentation Time Stamp" from the MPEG system specification.

By structuring the protocol architecture in this way, RTP facilitates the ALF/JSCC design approach. Payload format specifications can be tailored for Internet transmission, effectively modifying the source-coding algorithm to match the channel (i.e., JSCC) and likewise modifying the ADU to match the underlying network (i.e., ALF).

4.4.1 RTP Video

One example of the ALF/JSCC design process is our formulation of the RTP-based Intra-H.261 packet video compression format. Our goal was to design a video compression algorithm for RTP that could cope with the burst erasures of Internet packet loss. As with TCP, separation does not work well here because burst erasures require many packets of delay in order to construct burst loss-resilient codes. Furthermore, a burst of lost packets can occur across many consecutive packets (i.e., when a link becomes congested) and protecting against this requires error control codes that span even larger time scales. Moreover, the dynamic variability of the underlying channel statistics (i.e., level of congestion) causes inefficiencies in our choice of error correcting codes at the source.

Before we propose a compression format that is jointly designed for a packet erasure channel, we first examine how a specific source-coding algorithm, H.261, is sensitive to burst losses. The fundamental challenge in adapting H.261 for packet transmission is its assumption that the underlying channel is a continuous serial bit stream with isolated and independent bit errors. As a result, the source-coding algorithm assumes that all bits reach the decoder intact with very high probability and thus that recovery procedures for missing data are rarely needed. When bits are lost or corrupted, the decoder must resynchronize its state along three dimensions:

- **Entropy-code Synchronization.** The coded symbol stream consists of entropy-coded variable length codewords. If the bitstream is fragmented within a codeword, the decoder cannot correctly interpret the remaining codewords. Furthermore, the syntax is context dependent — that is, the entropy codes at the current point depend strongly on how the bit stream was previously parsed. For example, the decoder uses one Huffman code to parse DCT coefficients and a different Huffman code to parse macroblock types.

- **Spatial Synchronization.** Even if the codec can resynchronize its entropy-decoding stage after a transmission error, spatially predicted state may remain corrupt because the predictor cannot subsume the lost data. For example, block addresses are differentially encoded. Hence, a loss of just one block address difference will corrupt the block address until it is reset at a synchronization point causing image blocks to be rendered into the wrong location within the frame.

- **Temporal Synchronization.** H.261's temporal prediction model assumes transmission errors are rare and therefore codes most block updates differentially (as they compress better than "intra-mode" updates). Consequently, lost packets result in lost differential updates, which cause reconstruction errors to persist in the decoder's prediction loop.

Because the H.261 source-coding algorithm assumes a reliable bit-oriented channel, the resynchronization mechanism is relatively simple. A frame is partitioned into a number of spatially disjoint units each called a "Group of Blocks", or GOB. Each GOB is prefixed by a special "start code" that is not a legal codeword and therefore cannot appear anywhere else in the bit stream. When the decoder loses synchronization, it scans for the next GOB start code to resume decoding (at the cost of discarding all intervening bits). Although a GOB boundary is a synchronization point for the entropy coder and spatial prediction state, it cannot resynchronize the temporal prediction state without loss of compression efficiency. Moreover, GOBs are coarse-grained objects within a frame and thus occur infrequently.

Nevertheless, we can adapt the H.261 resynchronization algorithm for packet transmission by simply aligning the start of each packet on a GOB boundary. Because a GOB provides a well-defined starting point where decoder state is synchronized to well-known initial values, we can decode a given GOB independent of other GOBs, and in particular, we can decode it independent of packet loss in other GOBs. This approach was taken in early versions of Turletti and Huitema's "RTP Payload Format Specification for H.261" [166], but it had several shortcomings:

(i) A GOB does not necessarily fit within a packet and often must be fragmented across multiple packets. The payload format specified mechanisms for fragmentation but because these mechanisms did not explicitly account for the underlying bit stream semantics, a lost fragment could cause later packets to be undecodable and thus useless.

(ii) Not only might a GOB be too large for a packet, but the variability in GOB sizes can cause an inefficient packing of GOBs into packets. That is, some packets may be full while others are almost empty thereby increasing the number of packets needed to encode a frame (consequently increasing the per-packet overhead).

(iii) Under packet loss, the coding scheme suffered from artifacts caused by mismatched temporal prediction state.

To overcome these limitations, we adopt a JSCC/ALF approach and propose modifications to the source-coding algorithm to better match the packet transmission model. From the source-coding perspective, our changes introduce overhead that increases the bit rate for a given level of quality, but the advantages they provide in robustness to packet loss outweigh this overhead. In the next two sections, we describe a macroblock-based fragmentation that overcomes the limitations of GOB-oriented processing and a block-based conditional replenishment that overcomes the limitation of the temporal prediction model.

4.4.2 Macroblock-based Packet Fragmentation

To solve the spatial synchronization problem, we flatten the coding hierarchy. Rather than decompose a frame into GOBs, which in turn are decomposed into macroblocks, we adopt a single level decomposition where a frame is composed of macroblocks[1]. Further, we compactly represent the spatial prediction in a header at the start of each packet, exploiting the fact that a packet arrival implies that all of the bits of that packet have arrived completely intact.

Because macroblocks are small compared to packets, we have fine-grained control over formatting them into packets and thus can map ADUs explicitly onto network packets by ending a packet on an arbitrary block boundary. Unfortunately, a source does not in general know the maximum packet size supported by the underlying networks. If the source uses packets that are too large, then the Internet Protocol network layer will fragment the datagrams into smaller packets and confound our one-to-one mapping of ADU onto packet. For unicast transmission, a source can use "Path MTU Discovery" [103, 126] to learn the maximum transmission unit allowed along a certain unicast path through the network. (Note that this is another example of joint source channel coding where we are using network properties to control the source-coder.) But for multicast transmission, we have no such path discovery mechanism and rely on a rule of thumb for choosing packet sizes: we choose a size that is small enough to be carried by all of the prevalent physical-layer network technologies and is simultaneously large enough to sufficiently amortize the cost of packet headers. In the MBone, we currently use packet sizes of roughly 1000 bytes.

To summarize, macroblock-based framing of compressed bits into packets performs well because:

- a packet-switched network already includes overhead to identify packet boundaries, so there is no need to include resynchronization codes in the source-coding syntax;
- a packet identifies an explicit boundary that facilitates decoder resynchronization by including full decoder state at the start of each packet (e.g., first block number, spatial DC predictor, motion vector predictor if in use, and so forth);
- the overhead of the decoder state is amortized by sufficiently large packet payloads; and,
- every received packet can be decoded even when other packets in the stream are lost.

4.4.3 Conditional Replenishment

We solve the temporal synchronization problem with a simple temporal compression model called *conditional replenishment*. In block-based conditional replenishment, the input image is gridded into small blocks (e.g., 8x8 or 16x16 pixels) and only the blocks that change in each new frame are encoded and transmitted. To avoid dependencies between the new block and previously transmitted blocks, we intra-code each block update rather than code a residual prediction error.

[1] The macroblock-based fragmentation strategy arose from a series of email exchanges with Atanu Ghosh and Mark Handley in the summer of 1994.

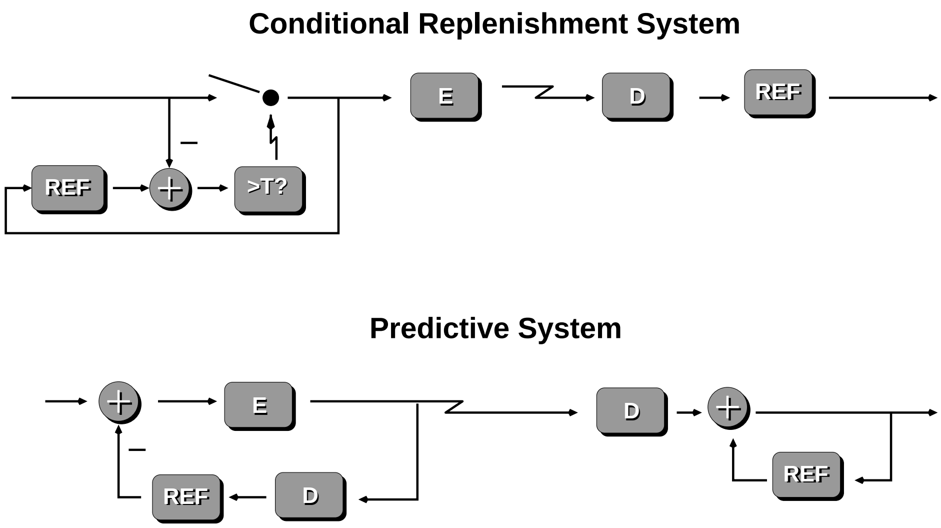


Figure 4.2: **Temporal Compression Models.** A conditional replenishment system encodes and transmits blocks as independent units, while a predictive system encodes and transmits the residual error between a prediction and the input signal.

Figure 4.2 depicts a block diagram for the conditional replenishment algorithm. The encoder maintains a reference frame of transmitted blocks. For each new block, a distance between the reference block and the new block is computed. If the distance is above a threshold, the block is encoded and transmitted across the network. At each receiver, the new block is decoded and placed in a reconstruction buffer for rendering and eventual display.

In contrast, compression algorithms like H.261 (or MPEG and H.263) employ temporal prediction to achieve higher compression performance. To first order, these schemes compute a difference between the current block and the previously transmitted block and code this "prediction error". If the block does not change much, then the difference signal has low energy and can be substantially compressed. Often, the encoder compensates for camera pan and scene motion by sending a "motion vector" with each block that accounts for a spatial displacement between the current block and the reference frame at the decoder (a copy of which is maintained at the encoder).

While the compression performance of motion-compensated prediction exceeds that of conditional replenishment in the absence of packet loss, there are a number of significant advantages of conditional replenishment:

- **Reduced Complexity.** Because the encoder decides very early in the coding process not to code a block, many of the input blocks are simply skipped, thereby saving computational resources. Moreover, because the encoder does not form a prediction signal, there is no need to run a (partial) copy of the decoder at the encoder.

- **Loss Resilience.** Coding block differences rather than the blocks themselves substantially amplifies the adverse effects of packet loss. When a loss occurs, the resulting error persists in the decoder's prediction loop until the coding process is reset with an "intra-mode" update. That is, the loss of a single differential update causes the error to propagate from frame to frame until the decoder resynchronizes.

In H.261, for example, these updates can be very infrequent—as little as once every 132 frames. As a result, packet loss causes persistent corruption of the decoded image sequence. Alternatively, the use of "leaky prediction" lessens the impact of errors but incurs increased complexity and slower recovery [128, Ch. 5].

- **Decoupled Decoder State.** In the temporal prediction model, there is a tight coupling between the prediction state at the encoder and that at the decoder. But in a heterogeneous multicast environment, each decoder might receive a different level of quality and hence have a different reference state from which to construct the prediction. Since the "base layer" state is common across all receivers, the encoder can use it to perform the prediction. But in practice, the base layer provides inadequate conditional information to improve compression performance significantly across all of the layers. In contrast, conditional replenishment gives the advantage of temporal block suppression across all layers without relying on a matched decoder state.

- **Compute-scalable Decoding.** Heterogeneity exists not only in the network but also across end-systems, where some receivers might be outdated workstations while others are high-performance PCs. Consequently, in addition to packet loss in the network, messages can be lost in the end-system when the decoder cannot keep up with a high-rate incoming bit stream. In this case, the decoder should gracefully adapt by trading off reconstruction quality to shed work [34, 51]. However, such adaptation is difficult under the temporal prediction model because the decoder must fully decode *all* differential updates to maintain a consistent prediction state. In contrast, with conditional replenishment, compute-scalability is both feasible and simple. The decoder simply collapses multiple frame updates by discarding all but the most recent compressed representation of each block.

  Moreover, conditional replenishment does not suffer from the well-known *decoder drift* effect. In predictive algorithms, the decoder's prediction state can gradually drift away from the encoder's because of numerical inconsistencies in the encoder and decoder implementations. (To limit the degree of decoder drift, compression specifications typically define the tolerances and the time extent between synchronization points.) On the other hand, conditional replenishment accommodates compute-scalable algorithms at both the decoder and encoder because there is no prediction loop to cause decoder drift. Here we can exploit numerical approximations to trade off reconstruction quality for run-time performance. For example, the inverse DCT could be replaced by an approximate algorithm that runs faster at the expense of decreased accuracy [102]. Likewise, the degree of quantization applied to the DCT coefficients can be dynamically manipulated to meet a computation budget [110].

- **Self-correlated Updates.** The update heuristic that transmits only blocks that change works well in practice because block updates are "self-correlated". If a certain block is transmitted because of motion in the scene, then that same block will likely be transmitted again in the next frame because of the spatial locality of motion. Thus a block update that is lost in a dropped packet is often soon thereafter retransmitted and recovered as part of the natural replenishment process.

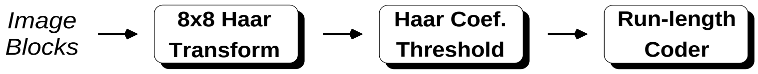
- **Static-background Video.** Finally, the class of video currently sent over the Internet is primarily teleseminars and video conferences where large static backgrounds often dominate the scene and conditional replenishment is highly effective.

For these reasons, we sacrifice the compression advantage of temporal prediction for the simplicity and practical advantages of conditional replenishment. In short, our compression algorithm exploits temporal redundancy only through conditional replenishment.

Although we believe that conditional replenishment provides a reasonable tradeoff between compression efficiency and loss resilience, it is an extreme point in the continuum between full intra-coding and very infrequent inter-coding of block updates. If there is no packet loss in the network, then it is advantageous to use temporally predictive coding to improve compression efficiency. But under high loss, intra-only coding achieves higher overall performance. Thus, a scheme that monitors conditions in the network and adjusts its coding algorithm reactively (i.e., JSCC) will likely perform better. Turletti and Huitema proposed such a loss-adaptive source-coding technique where the interval of intra-mode block updates (i.e., the amount of rate allocated to redundancy) is controlled by observations of the network [167]. Although this style of adaptation works well for unicast transmission, the update interval is fixed at the source and cannot accommodate multiple receivers with heterogeneous packet loss rates.

4.5 A Case Study: nv and Intra-H.261

A very simple approach for Internet video that uses both macroblock-based fragmentation and conditional replenishment first appeared in the Xerox PARC "Network Video" tool, *nv*. The high-level compression model utilized by *nv* is decomposed as follows [62]:



Here, 8x8 image blocks from the conditional replenishment stage are transformed using a Haar wavelet decomposition. A threshold is then applied to each coefficient and coefficients with magnitude below the threshold are set to zero. This process creates runs of zeros, which are run-length coded in the last stage. Finally, the coded blocks are formatted into packets and transmitted on the network. Since the Haar transform requires only additions and subtractions and the threshold step requires only two conditionals, the algorithm has very low computational complexity. Unfortunately, compression performance suffers because the Haar transform provides relatively poor energy compaction of the signal [93] and the entropy coder is based exclusively on fixed size run-length codes.

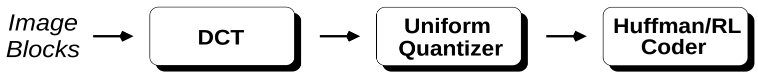
The performance of the *nv* coder can be substantially improved with the following changes:

(i) Replace the Haar transform with the Discrete Cosine Transform (DCT), which has good energy compaction for images [93].

(ii) Replace the coefficient threshold stage with a uniform quantizer to reduce the entropy of quantized coefficients.

(iii) Follow the run-length coder with a Huffman coder to further compress the symbol stream.

The modified encoder structure then becomes:



Intra-H.261. These changes amount to applying the compression advantages of the DCT-based approaches like the H.261 format described above to the *nv* coding algorithm. Since the scheme so closely matches H.261, we created an H.261 variant that leverages off these ideas from *nv*. In fact, it turns out that one can get the advantages of aggressive conditional replenishment and intra-coded blocks with a fully-compliant H.261 syntax.

We use this technique, which we call "Intra-H.261"[2], in *vic*. Intra-H.261 uses only intra-mode H.261 macroblock types and uses macroblock addressing to skip over unreplenished blocks. Hence, we implement our macroblock-based fragmentation and conditional replenishment coding models simply by constraining the H.261 standard. Because the encoder uses only a small, simple subset of H.261, the implementation is straightforward (a few hundred lines of C++).

We achieve good computational performance by folding quantization into the DCT computation and by using an efficient 80-multiply 8x8 DCT [135]. We experimented with several vector-radix DCTs [29] but found that the separable row-column approach, despite having asymptotically higher complexity, performed better in the 8x8 case because of reduced memory traffic. Furthermore, because Intra-H.261 never sends inter-coded blocks, the algorithm avoids computing a prediction error signal. This prunes much of the computation because it eliminates the need to run a (partial) copy of the decoder within the encoder (i.e., it eliminates an inverse quantization and inverse DCT of every encoded block).

Since its release within *vic* in November 1994, the Intra-H.261 coding format has become the dominant compression format for MBone video transmissions and has been incorporated into several commercial products that employ Internet video transmission. In the summer of 1995, the RTP/H.261 packet format was modified to include fields for the spatial prediction state required by macroblock-based fragmentation. These changes were based on our experiences with Intra-H.261 in combination with the pioneering work on the original adaptation of H.261 for Internet transmission in *ivs* [167]. The resulting payload format and fragmentation scheme is detailed in [167].

4.6 Performance Results

Because the Intra-H.261 and *nv* compression schemes use similar conditional replenishment algorithms, we can evaluate their relative compression performance by ignoring the temporal dimension and comparing only their 2D image compression performance. Figure 4.3 shows the relative performance of the two approaches for the canonical 8-bit, 512x512 grayscale "Lena" image. Both encoders were modified to omit block-addressing codes; note that this allows the H.261 encoder to operate on a non-standard image size. These modifications have little impact on the results since block-addressing accounts for only a small fraction of the bit rate.

The figure shows the peak signal-to-noise ratio (PSNR) plotted against rate (in bits per pixel). Multiple points were obtained by varying the H.261 scalar quantizer and the *nv* dead-zone threshold. Note that the *nv* algorithm was intended to operate with a non-configurable, fixed threshold, but we explored other thresholds to complete a rate-distortion curve.

As seen in the graph, H.261 consistently outperforms the *nv* coder by 6-7dB. Since transmissions in the MBone are typically rate-limited, we can alternatively consider a fixed distortion and compare the corresponding bit rates. From this perspective, the *nv* bit rate is two to three times that of H.261 for the same level of quality. For a rate-limited transmission, this translates into a two to threefold reduction in frame rate.

In addition to improving upon the compression performance of the *nv* format, we improved upon the run-time performance of the H.261 coder in *ivs*. To assess the degree of this improvement, we tested *vic* version 2.6.2 and *ivs* version 3.4 on an SGI Indy (133MHz MIPS R4600SC) with the built-in VINO video device. Both applications were compiled with gcc 2.6.2 using the -O2 flag. We ran the programs individually, giving each application a CIF-sized high-motion input and a low-motion input. In order to measure the

[2] Although the name "Intra-H.261" implies that the entirety of every frame of video is coded, the scheme in fact uses conditional replenishment to skip over blocks that do not change. We regret changing the name from its earlier, superior form: *Robust-H.261*.

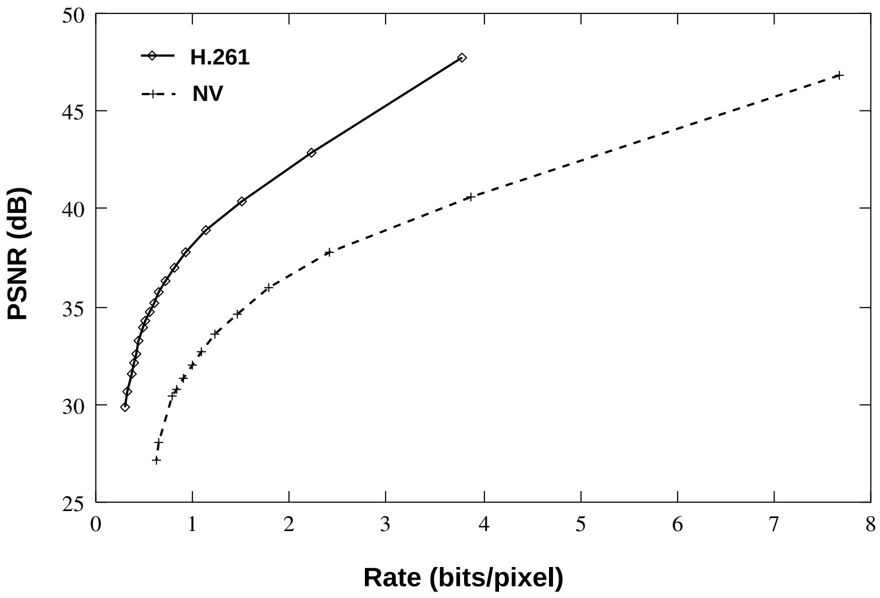


Figure 4.3: **Compression Performance of Intra-H.261 and nv.** Intra-H.261 provides a two to threefold improvement in compression gain over the *nv* encoder, or about 6 dB of PSNR.

	high-motion	*low-motion*	*cpu. util.*
ivs	3.5 f/s	20 f/s	100%
vic	8.5 f/s	30 f/s	40%

Table 4.1: H.261 Run-time Comparison

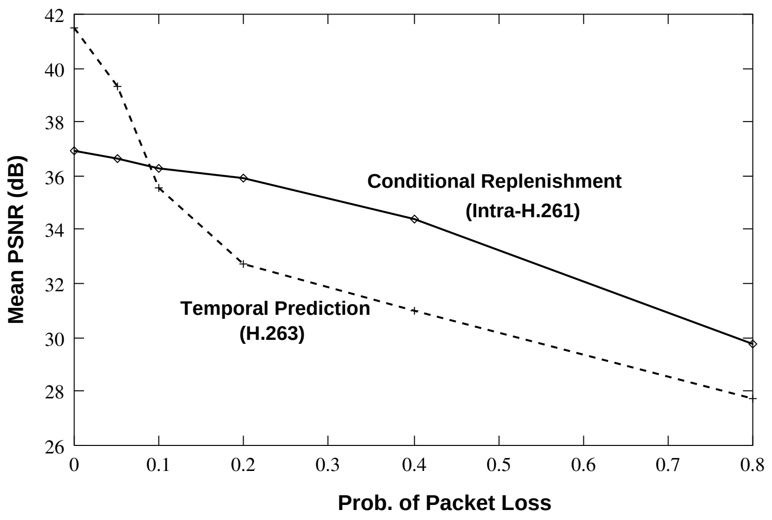


Figure 4.4: **Temporal Prediction Loss Resilience.** Conditional replenishment is more robust to packet loss compared to traditional temporal compression models based on predictive coding. As the packet loss rate increases, the reconstructed quality of the conditional replenishment stream degrades more slowly than that of temporal prediction.

maximum sustainable compression rate, we disabled the bandwidth controls in both tools and ran the test on an unloaded machine. We measured the resulting frame rates by decoding the streams on separate machines. The results are given in Table 4.1. For the high-motion case, almost all the blocks in each frame are coded, so this gives a worst-case performance bound. For the low-motion case, *vic* ran at the full NTSC frame rate and thus was operating below 100% utilization of the CPU. We therefore measured the utilization and found it to be 40%, which adjusts the 30 f/s measure to 75 f/s.

Conditional-Replenishment. To quantify the advantages of conditional replenishment over temporal prediction, we compared the relative performances of Intra-H.261 and of the newer ITU H.263 [172] standard in a simulation of different packet loss rates in the network. H.263 is widely regarded as the state of the art in low bit-rate video coding. We coded the standard Miss America grayscale test sequence using both Intra-H.261 and the Telenor implementation[3] of H.263. As described above, Intra-H.261 uses only conditional replenishment (i.e., all block updates are intra-coded); the H.263 codec uses only temporal prediction (i.e., all block updates other than those in the first frame are differentially coded). We simulated loss by fragmenting the bit stream into fixed size packets and marking packets "lost" according to independent Bernoulli trials with probability equal to the packet loss rate. The bit stream was decoded as normal, but any block update that was contained in a lost packet was discarded. The fragmentation process introduced no entropy or spatial synchronization errors and instead we assumed that for both cases synchronization is recovered using negligible side information provided on each packet boundary.

Figure 4.4 plots the average PSNR across the entire frame sequence versus the packet loss rate averaged over a number of simulation runs. We ran the simulation a sufficient number of times to reduce the confidence intervals to a negligible range. As expected, both curves decrease monotonically since reconstruction quality degrades with increasing packet loss. Beyond about 8% packet loss, the performance

[3] On-line at http://www.nta.no/brukere/DVC/.

of conditional replenishment exceeds temporal prediction. Moreover, the conditional replenishment curve decays much more slowly than the temporal prediction curve, indicating its higher resilience to packet loss.

4.7 Summary

In this chapter, we argued that the separation principle breaks down in Internet-like environments because interactive transmission requires tight delay constraints and burst erasures from packet losses on multiple time scales confounds forward error-control coding that assume bit-oriented errors. We developed several arguments and presented simulation results and performance measurements in support of JSCC and ALF. For certain applications and environments, namely video delivery over the Internet, we claim that designs based on JSCC and ALF have potentially greater performance than those based on the independent design of the compression and transmission components. To support this argument, we described our Intra-H.261 variant of the ITU H.261 standard that embodies ALF and JSCC through conditional replenishment and macroblock-based fragmentation.

Chapter 5

Receiver-driven Layered Multicast

Having established our network model based on the IP Multicast protocol architecture and our system model based on application level framing (ALF) and joint source/channel coding (JSCC), we now turn to a detailed discussion of the core contribution of this thesis: our layered compression and transmission architecture for multicast video. In the next two chapters, we present an ALF/JSCC-based system composed of a layered transmission protocol (described in this chapter) combined with a layered compression algorithm (described in the next chapter).

Although we proposed an ALF- and JSCC-based design methodology using the examples of conditional replenishment and Intra-H.261 in the previous chapter, we did not explicitly address the problem of heterogeneous video transmission nor did we explore techniques for adapting the source coding algorithm to congestion in the network. As described in Chapter 1, a typical configuration for multicast video consists of a heterogeneous set of hosts receiving video from some number of sources all within one multicast session. But because the receivers are connected to the network at different rates, the source cannot adjust its transmission to simultaneously satisfy the rate requirement of each receiver. We solve this problem by moving the burden of rate adaptation from the source to the receivers in a scheme we call Receiver-driven Layered Multicast (RLM). Under RLM, a layered signal is transmitted over multiple IP Multicast groups — each layer of a hierarchical media stream is distributed on an individual IP multicast group — and receivers adjust their rate of reception by controlling the number of groups they subscribe to. Although we have presented an architecture for delivering multiple levels of quality using multiple network channels to heterogeneous sets of receivers, we have not specified the mechanisms for the system to decide how many flows should be forwarded across each link or how the receivers collectively determine the "best" number of layers that each locally receives.

In this chapter we develop the distributed algorithm and the network protocol that comprise RLM. Under RLM, each receiver individually adapts to observed network performance by adjusting its level of subscription within the overall layered multicast group structure. Moreover, all the members in a session share control information across the group to improve the convergence rate of the adaptation algorithm.

The remainder of this chapter is organized as follows. In the next section we give a detailed overview of the RLM protocol, the protocol state machine run at each receiver, and the shared learning algorithm that enhances the protocol's scalability. We then describe an extension to the IP Multicast architecture called "Tentative-join with Failure Notification" (TFN) that complements RLM to strengthen its adaptation robustness. Next we present a large number of simulation results to show that our protocol's performance scales gracefully under a number of typical network configurations. Finally, we explore the interaction of TCP and RLM in simple configuration and compare the performance of RLM with that of the *ivs* congestion control scheme.

5.1 The RLM Protocol

Building on the best-effort IP-Multicast network model described in Chapter 3, we now describe RLM at a high level to develop intuition for the protocol before discussing the low-level details. In RLM, a session is comprised of a number of active sources transmitting layered streams to a number of receivers distributed throughout the network. Since bandwidth between a given receiver to any particular active source may vary, RLM treats each source independently and runs a separate instance of the adaptation protocol for each incoming stream.

In effect, the source takes no active role in the protocol: it simply transmits each layer of its signal on a separate multicast group. The key protocol machinery is run at each receiver, where adaptation is carried out by joining and leaving groups. Conceptually, each receiver runs the following simple control loop:

- on congestion, drop a layer;
- on spare capacity, add a layer.

Under this scheme, a receiver searches for the optimal *level of subscription* much as a TCP source searches for the bottleneck transmission rate with the slow-start congestion avoidance algorithm [87]. The receiver adds layers until congestion occurs and backs off to an operating point below this bottleneck rate.

Figure 5.1 illustrates the fundamental mechanism underlying RLM. Suppose source S transmits three layers of video to receivers R_1, R_2, and R_3. Because the S/R_1 path has high capacity, R_1 can successfully subscribe to all three layers and receive the highest quality signal. However, if either R_2 or R_3 try to subscribe to the third layer, the 512 kb/s link becomes congested and packets are dropped. Both receivers react to this congestion by dropping layer 3, prompting the network to prune the unwanted layer from the 512 kb/s link. Finally, because of the limited capacity of the 128 kb/s link, R_3 drops down to just a single layer. In effect the distribution trees for each layer are implicitly defined as a side effect of receiver adaptation.

5.1.1 Capacity Inference

To drive the adaptation, a receiver must determine if its current level of subscription is too high or too low. By definition, the level of subscription is too high if it causes congestion. This is easy to detect because congestion is expressed explicitly in the data stream through lost packets and degraded quality. On the other hand, when the level of subscription is too low, there is no equivalent signal — the system continues to operate at its current level of performance. We must therefore rely on some other mechanism to provide this feedback.

One source for this feedback might be to monitor link utilization and explicitly notify end-systems when capacity becomes available, but this requires new mechanism in the network that renders deployment difficult. The approach we adopt in RLM is to carry out active experiments by spontaneously adding layers at "well chosen" times. We call this spontaneous subscription to the next layer in the hierarchy a *join-experiment*. If a join-experiment causes congestion, the receiver quickly drops the offending layer. If a join-experiment is successful (i.e., no congestion occurs), then the receiver is one step closer to the optimal operating point.

5.1.2 RLM Adaptation

Unfortunately, join-experiments cause transient congestion that can impact the quality of the delivered signal. Therefore, we need to minimize the frequency and duration of join-experiments without im-

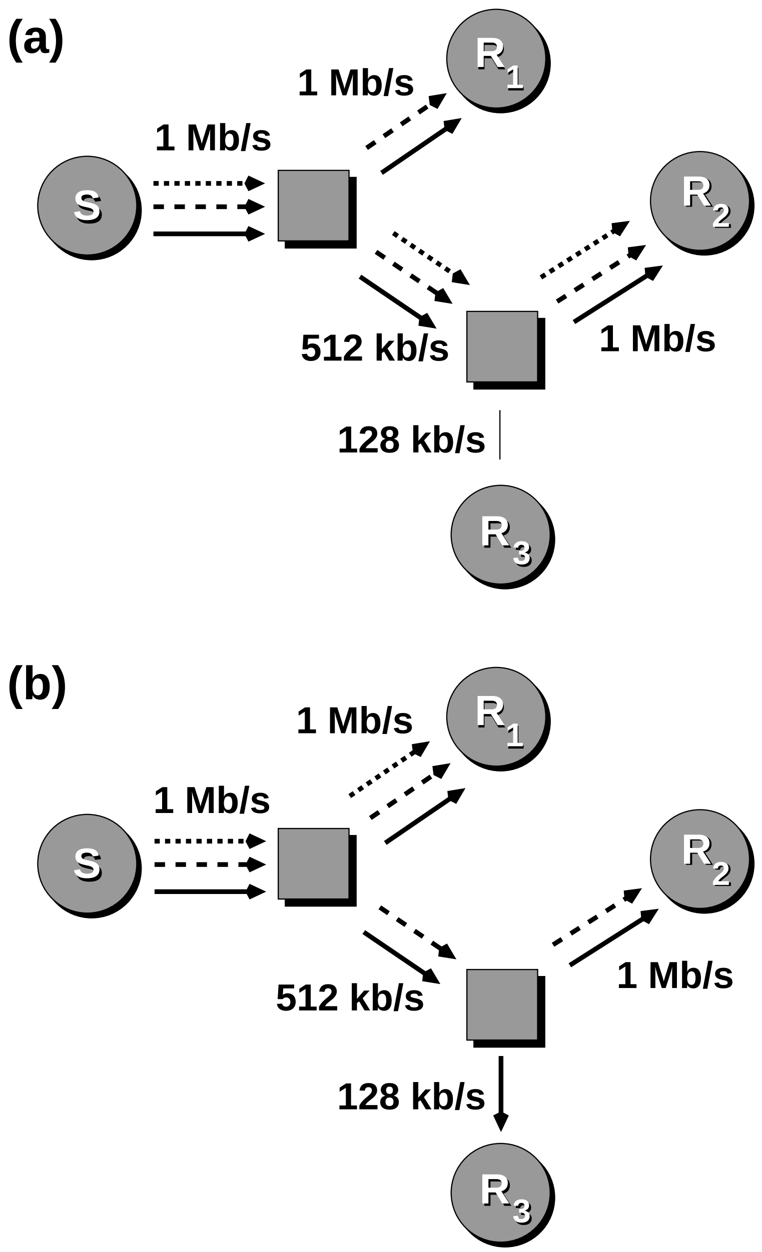


Figure 5.1: **End-to-end Adaptation.** Receivers join and leave multicast groups at will. The network forwards traffic only along paths that have downstream receivers. In this way, receivers define multicast distribution trees implicitly through their locally advertised interest. A three-layer signal is illustrated by the solid, dashed, and dotted arrows, traversing high-speed (1 Mb/s), medium-speed (512 kb/s), and low-speed (128 kb/s) links. In (a), we assume that the 512 kb/s is oversubscribed and congested. Receiver R_2 detects the congestion and reacts by dropping the dotted layer. Likewise, receiver R_3 eventually joins just the solid layer. These events lead to the configuration in (b).

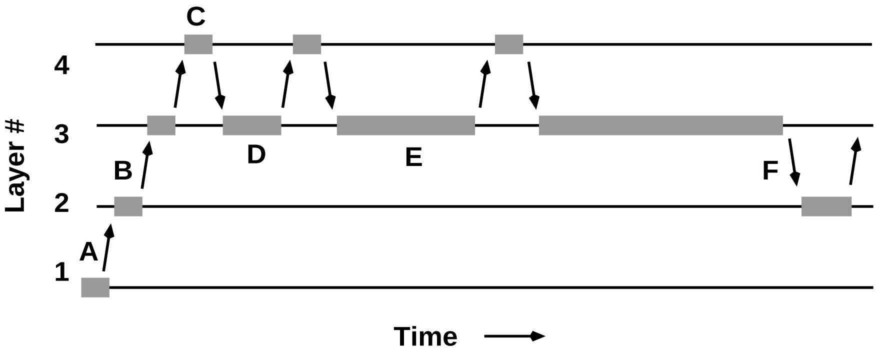


Figure 5.2: **An RLM "Sample Path".** This diagram illustrates the basic adaptation strategy from the perspective of a given receiver. Initially, the receiver joins the base layer and gradually adds layers until the network becomes congested (C). Here, the receiver drops the problematic layer and scales back its join-experiment rate for that level of subscription.

pacting the algorithm's convergence rate or its ability to track changing network conditions. This is done through a learning algorithm, where over time, each receiver determines the level of subscription that causes congestion. By doing join-experiments infrequently when they are likely to fail, but readily when they are likely to succeed, we reduce the impact of the experiments. We implement this learning strategy by managing a separate *join-timer* for each level of subscription and applying exponential backoff to problematic layers.

Figure 5.2 illustrates the exponential backoff strategy from the perspective of a single host receiving up to four layers. Initially, the receiver subscribes to layer 1 and sets a join-timer (A). At this point, the timer duration is short because the layer has not yet proven problematic. Once the join-timer expires, the receiver subscribes to layer 2 and sets another join-timer (B). Again, the timer is short and layer 3 is soon added. The process repeats to layer 4, but at this point, we assume that congestion occurs (C). A queue then builds up and causes packet loss. Once the receiver detects these lost packets, it drops back to layer 3. The layer 3 join-timer is then multiplicatively increased and another timeout is scheduled (D). Again, the process repeats, congestion is encountered, and the join-timer is further increased (E). Later, unrelated transient congestion provokes the receiver to drop down to layer 2 (F). At this point, because the layer 3 join-timer was never increased in response to congestion, the layer is quickly reinstated.

In order to properly correlate a join-experiment with its outcome, we must know how long it takes for a local layer change to be fully established in the network and for the resulting impact to be detected back at the receiver. We call this time interval the *detection-time*. If a join-experiment lasts longer than the detection-time without congestion occurring, then we deem the experiment successful. On the other hand, if congestion occurs within the detection-time interval, we assume the experiment failed and increase the join-timer for that layer. Because the detection-time is unknown and highly variable, we estimate it and its variance adaptively. We initialize our estimator (mean and deviation) with a conservative (i.e., large) value, and adapt it using failed join-experiments. That is, when an experiment fails, we update our estimator with the time interval between the start of the experiment and the onset of congestion.

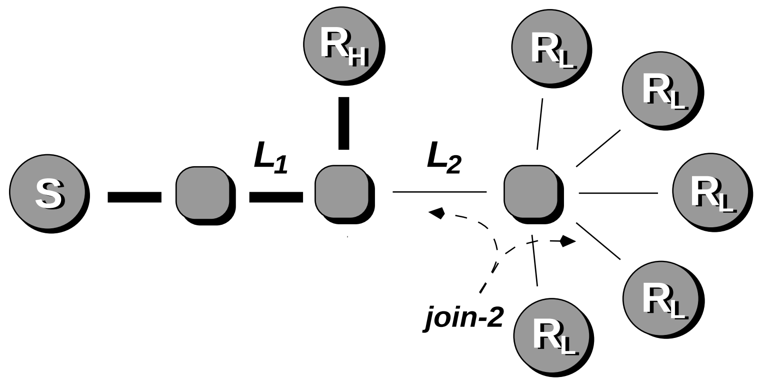


Figure 5.3: **Shared Learning.** Upon conducting a join-experiment, the receiver R_L at the bottom of this diagram multicasts an explicit join-2 message to the entire group. Thus when link L_2 becomes congested as a result of the join-experiment, all downstream receivers simultaneously learn that layer 2 is problematic and hence adjust their layer 2 join timers.

5.1.3 Shared Learning

If each receiver carries out the above adaptation algorithm independently, the system would scale poorly. As the session membership grows, the aggregate frequency of join-experiments increases; hence, the fraction of time the network is congested due to join-experiments increases. Moreover, measurement noise increases because experiments tend to interfere with each other. For example, if one receiver is conducting an experiment on layer 2 and another begins an experiment on layer 4 that causes congestion, then the first receiver can misinterpret the congestion and mistakenly back off its layer 2 join-timer.

We can avoid these problems by scaling down the individual join-experiment rates in proportion to the overall group size. In other words, we can fix the aggregate join-experiment rate independent of session size much as RTCP scales back its control message rate in proportion to the group size [153]. However, reducing the experiment rate in this manner decreases the learning rate. For large groups, the algorithm will take too long to converge.

Our solution is “shared learning”: Before a receiver conducts a join-experiment, it notifies the entire group by multicasting a message identifying the experimental layer. Thus all receivers can learn from other receivers’ failed join-experiments. For example, Figure 5.3 shows a topology with a single source, one receiver R_H situated along a high-speed path (denoted by the thick lines) and a set of receivers, each labeled R_L, situated at the far end of a low-rate link. Suppose a low-rate receiver decides to conduct a join-experiment on layer 2. It broadcasts a join-2 message to the group and joins the layer 2 multicast group. As a result, link L_2 becomes oversubscribed and congestion results, causing packets to be dropped indiscriminately across both layers. At this point, all of the R_L receivers detect the congestion and since they know a layer 2 experiment is in progress, they all scale back their layer 2 join-timer. Thus all of the low-bandwidth receivers learn together that layer 2 is problematic. Each receiver need not run individual experiments to discover this on its own.

This learning process is conservative. Receivers make their decisions based on failed experiments not on successful experiments. Moreover, the success/failure decision is based on local observations, not on a global outcome. That is, each receiver decides whether the experiment succeeds based on the network conditions on the path from the source to that receiver, entirely independent of the receiver that instantiated the join-experiment. Hence, a given experiment may succeed for some receivers but fail for others.

Even though the shared learning process enhances the protocol's scalability by reducing convergence time, overlapped experiments can still adversely impact the learning rate. But because receivers explicitly announce the start of each experiment, the probability that an experiment overlaps with another can be substantially reduced by suppressing the start of a new experiment when one is outstanding. For example, if in Figure 5.3 receiver R_H decides to carry out a join-4 experiment that causes congestion on link L_1, then the low-rate receivers can misinterpret this as a failed join-2 experiment. But because R_H sees the explicit join-2 announcement, it will suppress the join-4 experiment and thereby limit the interference. Note that this exchange of information is merely an optimization. If the announcement packet is lost, the algorithm still works albeit with potentially reduced performance.

The receivers can also use shared learning to collectively reduce the aggregate frequency of experiments in the steady state. If a receiver notices the failure of a join-experiment conducted by some other member in the group, it can presume that this same experiment would locally fail. Accordingly, whenever a receiver observes a failed experiment for a layer on which it is running a join-timer, it cancels existing timer and reschedules a new one. As a result, the group as a whole conducts experiments at an overall rate determined by the maximum join-timer parameter. For large groups, this interval tends to the minimum of the random join-timer interval[1].

Because the shared learning process determines what does *not* work rather than what does work, each receiver can advance its level of subscription only through actual join-experiments. If the suppression algorithm were completely exclusionary, then the convergence time could still be very large because each receiver would have to wait its turn to run an experiment. Instead, we allow experimental overlap if the pending level is the same as or less than the level in progress. This gives newer receivers with lower levels of subscription an opportunity to conduct experiments in the presence of a large population of established receivers at higher levels of subscription. Although this mechanism allows experimental overlap, a receiver that causes an overlap can condition its response accordingly by reacting more conservatively than in the non-overlapped case. The intuition behind this scheme is that high-layer receivers allow low-layer receivers to quickly adapt to their stable level of subscription. As the low-layer receivers adapt, their join-experiment frequency falls off and the high-layer receivers will again find idle periods in which to conduct join-experiments.

This technique for sharing information relies on the fact that the network signals congestion by dropping packets across all layers of the distribution. Under a priority-drop policy, receivers not subscribed to the experimental layer would not see packet loss and would not know the experiment failed. In short, a priority-drop policy interferes with the scalability of RLM, lending additional credence to the argument against drop priorities presented in §3.6.2.

5.1.4 Fairness

Although RLM receivers adapt locally to network capacity, the target operating point is not globally optimized. If multiple, simultaneous transmissions are sharing a single network, RLM apportions the bandwidth among each transmission in an ad hoc fashion. As described in §3.6.4, it is not generally possible to achieve a "fair" allocation of bandwidth without some additional machinery in the network, even if all the end-nodes cooperate [92]. Even if the bandwidth allocation were fair, the aggregate system performance, as measured by the sum of distortions at each receiver, would not be optimal. As shown in [170], minimization of the total distortion in general requires an exchange of information among receivers.

[1] If X_k is a continuous random variable on (a, b), then $\liminf\{X_n\}$ is a.

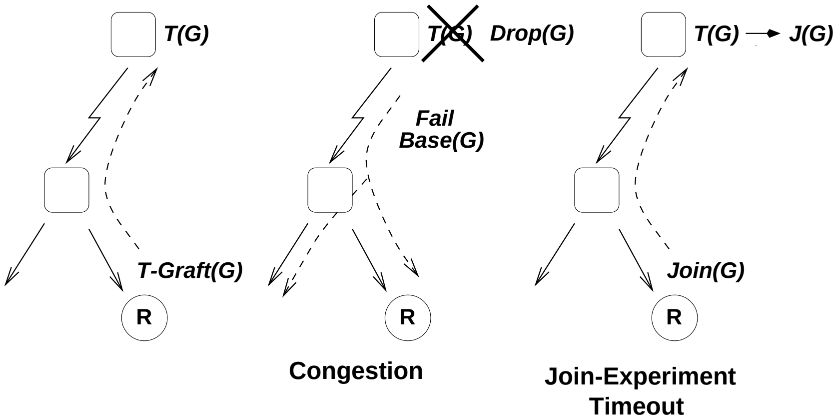


Figure 5.4: **Tentative Joins.** To "tentatively" join a group, a receiver sends a tentative-graft message up the tree to the graft point. If congestion occurs, the congested router drops the tentative group and multicasts a failure message down the corresponding sub-tree. If the join-timer expires, the receiver can "lock in" the group by issuing a normal join.

Most of our simulation results assume that routers drop packets on arrival when their queue is full. This widely deployed drop-tail discard policy is unfortunate because it delays the warning signal from the receivers until well after congestion has occurred and can cause phase synchronization effects that favor certain flows over others. RED gateways, on the other hand, react to incipient congestion by discarding packets at the onset of congestion (i.e., when the average queue size exceeds a threshold) [56]. Furthermore, RED's randomization tends to break synchronization effects that lead to grossly unfair performance, while its early reaction to congestion enhances RLM's performance because it allows receivers to react to congestion before the bottleneck link becomes fully saturated.

5.2 Tentative-join with Failure Notification

Although one of our key design goals in RLM is to avoid changes to the network, we can achieve improved performance if we relax this constraint. In this section, we explore an extension to the IP Multicast architecture called Tentative-join with Failure Notification (TFN) that strengthens RLM by providing explicit notification from network routers. We propose a new membership protocol message called a *tentative-graft*. The tentative-graft complements the normal graft operation with a mechanism that provides faster reaction to congestion, isolation of join-experiments, and constrains the result and impact of join-experiments to the topologically relevant set of receivers.

The tentative-join operation allows a receiver to convey to the network the experimental nature of its join operation. With this information, the network can react to congestion by dropping tentatively-joined multicast groups before dropping packets from other flows.

The semantics of the tentative-graft and its use in RLM are illustrated in Figure 5.4. When a receiver conducts a join-experiment, instead of joining the group outright, it *tentatively* joins the group. In response, the network propagates a tentative-graft (*T-Graft*) message for the group up the multicast tree toward the source. As the T-Graft message flows up the tree, each router installs the flow as normal but marks the group as tentative. Once the message reaches a router that has not pruned back the desired group, the T-Graft message is discarded and the flow is instantiated (as with a standard graft operation). As long as the tentative status of the group is in effect, a router will drop the corresponding group in response to congestion within the respective layer hierarchy. A receiver removes the tentative-status from the group by issuing a standard join operation (which generates a standard graft message).

Under congestion, the router not only drops the tentative group, but it simultaneously multicasts a failure message on the base-layer group to the multicast sub-tree downstream from the congested link. The address of a base-layer group can be either inferred from the address of the tentative group (i.e., by reserving a block of pre-defined addresses with a priori layer structure as described in § 3.5) or the address range can explicitly appear in the T-Graft message.

If congestion occurs during an RLM join-experiment, precisely those receivers who are impacted by the congestion are notified. Consequently, they can scale back their join timers for the corresponding layer in full confidence that the layer is problematic. Without TFN, RLM receivers can only infer that a join-experiment somewhere in the network is the cause of congestion. TFN removes all ambiguity about where the failure occurs and what group is responsible. Additionally, receivers that do not observe packet loss even when they are downstream of a failed experiment learn of the outcome through explicit notification.

If congestion does not occur during RLM join-experiment, then the experimenting receiver issues a standard join operation to remove the tentative status. This causes a standard graft message to flow up the multicast tree to the point where the group's status in no longer tentative. On each hop, the tentative status is lifted.

The net effect is that the tentative-graft in TFN provides a mechanism that improves RLM adaptation with:

- **Fast Reaction Time.** Since a router immediately reacts to the congestion precisely when and where it occurs, the reaction time is optimally fast. There is no sustained congestion while RLM decides that the path is in fact overloaded.

- **Session Isolation.** By isolating the join-experiments of one group from those of another, we minimize the congestive impact of RLM adaptation.

- **Topological Isolation.** By constraining notification messages to the region of the network that is affected, we minimize the measurement noise previously incurred when a receiver misinterprets unrelated congestion as a failed experiment within its session.

In addition, TFN gives us the benefit of drop priorities for layered transmission without providing incentive for receivers to run above the bottleneck capacity (as described in §3.6.2). If a receiver tentatively joins all layers, loss still occurs uniformly across the hierarchy.

Finally, we note that the tentative-graft messages can be deployed in a backward-compatible fashion. RLM receivers would still announce join-experiments but utilize explicit notification messages when and if they are present. Furthermore, routers can easily discover whether their neighbor supports the new protocol extension (by exchanging protocol version numbers), and if not, tentative-grafts could be transformed to normal grafts before propagating them. We can view the explicit notification as an optimization. Receivers always must react to sustained congestion but can respond more accurately and quickly to explicit notification when and where available [2].

[2]But with RED gateways, this time lag is smaller since congestion is signaled when the average queue size surpasses some small threshold.

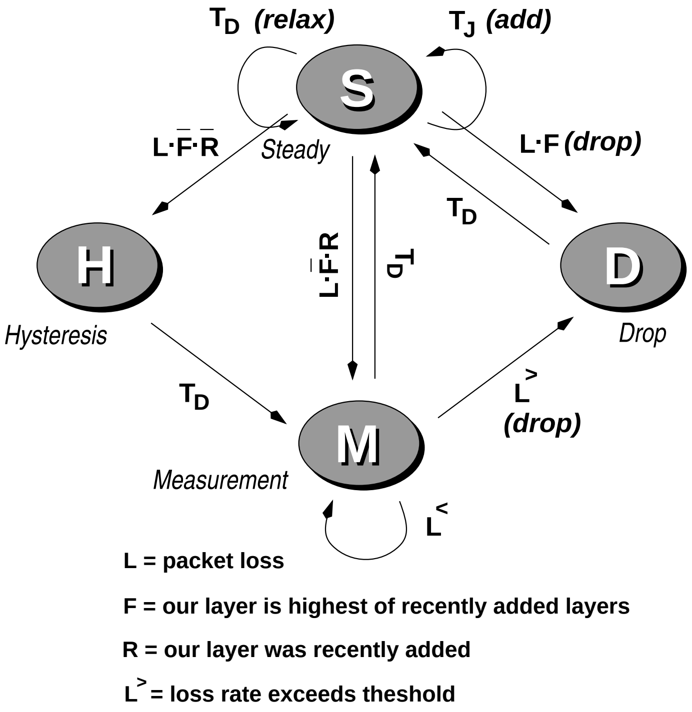


Figure 5.5: **RLM State Machine.** Each receiver maintains a simple four-state state machine. Timers, lost packets, and loss rate fluctuations cause state transitions, which in turn cause layers to be added and dropped.

Jamin proposed a somewhat similar mechanism called *preemptible service* [95, Ch. 11]. In this framework, layered transmission is combined with a measurement-based admission control system. When contention for the network exceeds its capacity, previously admitted low-priority flows are preempted to accommodate newly admitted high-priority flows.

5.3 Protocol Details

We now elaborate the protocol sketched in the previous section with the finite state machine depicted in Figure 5.5. There are four states: steady-state (S), hysteresis state (H), measurement state (M), and drop state (D). Each state transition is labeled with the reason for the transition, either packet loss or a timeout. Actions associated with a transition are indicated in parentheses.

Join-timers (T_J) are randomized to avoid protocol synchronization effects [57], while detection timers (T_D) are set to a weighted value of the detection-time estimator. The *add* action implies that we subscribe to the next layer in the multicast group hierarchy, while the *drop* action implies that we drop the current layer and multiplicatively increase the join-timer for that layer. The *relax* action implies that we multiplicatively decrease the join-timer for the current layer. When we scale back or relax the join-timer for a given layer to T, we set the join-timer in each of the layers above the new layer to the maximum of its current value and T and in each of the layers below the new layer to the minimum of its current value and T. This maintains the invariant that join-timers values increase with layer number. There are two types of loss actions: a fast reaction to a single packet loss (indicated by L) and a slower reaction to a sustained loss rate. The loss rate is measured with a short-term estimator and action is taken if the estimator exceeds a configured threshold (indicated by $L^{>}$).

In the S state, there is always a pending join-timer (unless the receiver is subscribed to all available layers). When the join-timer expires, we broadcast an explicit notification message to the group and add a layer. Upon reception of the join-experiment message, a receiver notes the experiment start time for that layer. In this way, we track the join-experiment activity at each layer and deem an experiment "in progress" if the time since the experiment started is less than

$$k_1 \hat{T}_D + k_2 \hat{\sigma}_D$$

where $\hat{T}_D$ is the detection-time estimator, $\hat{\sigma}_D$ is the detection-time sample mean-deviation, and k_1 and k_2 are design constants. If a lower layer join-experiment is in progress, we ignore the current join-timer and simply schedule a new one.

When loss occurs in the S state, the resulting action depends on the presence of active join-experiments. If there is a join-experiment in progress and our level of subscription corresponds to the highest-level join-experiment in progress, we infer that our join-experiment has failed, drop the offending layer, scale back the join-timer, and enter the D state. On the other hand, if we are locally conducting a join-experiment but a concurrent join-experiment is running at a higher layer, then it is likely that the higher layer experiment failed while ours did not but we cannot be certain. Hence, we enter the measurement state M to look for longer-term congestion before dropping our layer. Finally, if we were not conducting a join-experiment at all, we transition to the H state.

The H state provides hysteresis to absorb transient congestion periods. This prevents a receiver in steady-state from reacting to join-experiments that are carried out by other receivers in the network or to transient network congestion. Once the detection-timer expires, we assume that any transient join-experiment is finished and we transition to the measurement state and back to the S state after another detection time. If the congestion is long-term (e.g., because of new offered load), then once we enter the M state, the loss rate estimator ramps up, exceeds the threshold, and forces the current layer to be dropped.

When a layer is dropped in response to congestion, the receiver enters the D state, sets the detection-timer, and ignores losses until the detection-timer expires. This prevents the receiver from (over-)reacting to losses that are unrelated to its current level of subscription. Once the receiver has waited long enough, the incoming packet stream will reflect the new level of subscription and the receiver can take action on the subsequent quality.

5.3.1 Protocol State Maintenance

In addition to the current state identifier, the receiver control algorithm must maintain the current subscription level, the detection-time estimator, and the join-timers. This state information, along with several protocol design constants, is summarized in Table 5.1.

While the subscription level is trivial to maintain, the detection-time estimator and join-timers must be dynamically adapted to reflect changing network conditions. There are two operations performed on join-timers: backoff and relaxation. Call the mean of the join-timer for level-k, $\hat{T}_J^k$. Each timer interval is chosen randomly from a distribution parameterized by $\hat{T}_J^k$. When a join-experiment fails, the join-timer is multiplicatively increased:

$$\hat{T}_J^k \leftarrow \min(\alpha \hat{T}_J^k, T_J^{\max})$$

state	state identifier (S, H, M, D)
N	current level of subscription
$\hat{T}_J^k$	join-timer for level k
$\hat{T}_D$	detection-time sample mean
$\hat{\sigma}_D$	detection-time sample deviation
$T_J^{\min}$	minimum join-timer interval
$T_J^{\max}$	maximum join-timer interval
α	join-timer backoff constant
β	join-timer relaxation constant
k_1, k_2	detection-time estimator scaling term
g_1, g_2	detection-time estimator filter constants

Table 5.1: RLM State and Parameters

where $\alpha > 1$ is the backoff parameter and $T_J^{\max}$ is the maximum timeout. We clamp the backoff at a maximum to guarantee that a receiver will periodically probe for spare bandwidth. To scale to large session sizes, $T_J^{\max}$ is dynamically adjusted in proportion to the number of receivers. The number of receivers is in turn dynamically estimated through the exchange of session-wide control messages (e.g., as in RTCP [153]). Thus the aggregate join-experiment rate is fixed, independent of the session size, and packet loss induced by join-experiments does not increase with session size.

The join-timer undergoes relaxation in steady-state. The longer a receiver is in steady-state at some level, the more likely it is for that level to be stable. Thus the corresponding join-timer interval should be small. We adapt the join-timer by geometrically decreasing it at detection-timer intervals:

$$\hat{T}_J^k \leftarrow \max(\beta \hat{T}_J^k, T_J^{\min})$$

where $\beta < 1$ is the relaxation constant and $T_J^{\min}$ is the minimum join-timer interval.

While the join-timers are determined algorithmically, the detection-time estimate is derived directly from network measurements. The detection-time reflects the latency between the time at which a local action is carried out and the time at which the impact of that action is reflected back to the receiver. Note that this delay can be much larger than the time it takes for the network just to instantiate a new flow. If the new aggregate bandwidth exceeds the bottleneck link capacity by only a small amount, a long time may pass before a queue builds up and causes packet loss.

The detection-time estimate is computed by correlating failed join-experiment start times with the onset of congestion. Each time a join-experiment fails, the detection-time estimator assimilates the new latency measurement. The measurement, D_i, is passed through first-order low-pass filters with gains g_1, g_2:

$$\begin{aligned} \hat{\sigma}_D &\leftarrow (1 - g_2)\hat{\sigma}_D + g_2 |D_i - \hat{T}_D| \\ \hat{T}_D &\leftarrow (1 - g_1)\hat{T}_D + g_1 D_i \end{aligned}$$

5.4 Simulations

In this section, we present simulation results of several simple network topologies to explore the scalability of RLM. Although we investigate a number of simulation configurations and network topologies, our simulations do not prove that RLM is definitively scalable. Rather, they demonstrate that the scaling behavior is consistent with our intuition and show that, for simple scenarios, the protocol's performance is good. In a real network, performance will be affected by cross-traffic and competing groups, both of which add noise to the measurement process and introduce interactions that could potentially result in oscillatory behavior.

We implemented the RLM protocol described above in the UCB/LBNL network simulator *ns* [119]. Not only did this implementation serve as a framework for evaluating the protocol's performance, but the simulator provided feedback that was critical to the design process. *Ns* is an event-driven packet-level simulator controlled and configured via Tcl [131]. Shortest-path routes are computed for the input topology and multicast packets are routed via reverse-path forwarding. A flooding algorithm similar to Dense-mode Protocol Independent Multicast (PIM) [40] handles forwarding and pruning of multicast flows.

Layered video sources are modeled as a set of constant-bit rate (CBR) streams with fixed packet sizes. Packets are generated at times defined by the following law:

$$\begin{aligned} T_0 &= 0 \\ T_k &= T_{k-1} + \Delta + N_k, \ \ k > 0 \end{aligned}$$

where Δ is a fixed interval chosen to meet a target bit-rate and N_k is a zero-mean noise process that models variable coding delays ($\{N_k\}$ is i.i.d. uniform on $[-\Delta/2, \Delta/2]$). Unfortunately, this simple model fails to capture the burstiness of real video streams [65]. Because convergence in RLM relies on matching the layered rates to available capacity, smooth sources are well-behaved and this traffic model is overly optimistic. On the other hand, a bursty source can be smoothed out by applying rate-control through adaptive quantization at the cost of variable quality. A topic for future research is the degree to which RLM is amenable to bursty sources.

Before discussing the simulation results, we define the parameters we varied for the simulations and the metrics we used to evaluate the results. Variable parameters include network topology, link bandwidths and latencies, the number and rate of transmission layers, and the placement of senders and receivers. Fixed parameters include the routing discipline (drop-tail)[3], the router queue size (20 packets), and the packet size (1 KB). In all of our simulations, the link bandwidths are 510 kb/s, the traffic sources are modeled as a six-layer CBR stream at rates 32×2^m kb/s, $m = 0 \dots 5$, and the start-time of each receiver is randomly chosen uniformly on the interval $[30, 120]$ seconds. The protocol constants from Table 5.1 have the following values: $\alpha = 2$, $\beta = 3/4$, $k_1 = 1$, $k_2 = 2$, $g_1 = 0.25$, $g_2 = 0.25$, $T_J^{\min} = 5$ sec, $T_J^{\max} = 60$ sec. In practice, $T_J^{\max}$ would be much larger, but we set it to an artificially small value to obtain more frequent join-experiments and thereby reduce the amount of simulation time necessary to achieve negligible confidence intervals for our measurements. Each join-timer interval is chosen from $\lambda/2 + X$, where X is a random variable with density

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}/(1 - e^{-4\lambda^2}) & 0 \le x \le 4\lambda \\ 0 & \text{otherwise} \end{cases}$$

and $\lambda = \hat{T}_J^k$. These protocol parameters were chosen heuristically based on experimentation with and intuition about the protocol. An exploration of the protocol's sensitivity to these parameters as well as the development of analytic models that shed light on parameter tuning are both areas of future work.

[3] Drop-tail queues have been shown to contribute to protocol synchronization effects that can confound end-to-end performance [55]. In our simulations, drop-tail routers do not pose such problems because we optimistically inject randomness into the source model. We could achieve this same end using non-randomized sources in tandem with random-drop gateways.

5.4.1 Evaluation Metrics

In our layered multicast transmission scheme, a traditional metric like aggregate throughput is not well defined because each user might receive a different bandwidth and experience different loss rates. Performance not only depends on aggregate metrics like overall loss rate, but also on the stability of the system and the time scales over which events occur. Moreover, we must separate transient behavior from long-term behavior because an aggregate loss rate can be made arbitrarily good by letting the simulation run arbitrarily long after reaching stability.

To address these issues, we rely primarily on two metrics that (approximately) reflect the perceived quality of a real-time, loss-tolerant multimedia stream at the receiver. These two key metrics characterize the duration and frequency of "congestion periods". As a result of the probing mechanism in RLM, packet loss occurs in clusters. When no receiver is actively running a join-experiment, the network operates smoothly. But when a receiver probes for spare bandwidth, the network becomes momentarily congested causing the instantaneous packet loss rate to increase dramatically. The net effect is long periods without packet loss punctuated by short periods with very high loss rates. In short, the delivered quality of the video depends on the duration and frequency of these congestion periods.

There is no definitive way to decide when one congestion period ends and another begins. We could use join-experiment boundaries to determine congestion periods, but it would be difficult to discern when one join-experiment ends and another begins when experiments overlap or are closely spaced. However the individual cause of congestion is irrelevant — what matters is that congestion itself exists. Hence, we instead rely on a technique that identifies clusters of packet loss, i.e., congestion periods, simply by monitoring the incoming packet process at an individual receiver.

The performance evaluation literature contains many techniques for automatically identifying clusters in data sets [94], but we adopt the following very simple heuristic based on perceptual factors and the time scales on which RLM operates. We define a *loss event* to be the time interval that starts at a packet arrival just before a dropped packet and ends when a subsequent packet arrives. We further define a *congestion event* to be the union of loss events that are separated in time by less than some threshold T. Finally we define a *congestion period* as the start time of a congestion event's earliest loss event and the end time of the congestion event's last loss event. In our performance evaluation, we set $T = 2$ seconds.

Having defined a congestion period, we can now characterize performance based on duration and rate of congestion periods. We denote the congestion period duration by CPD and the congestion period interval by CPI. In our experiments, we typically run a given simulation some number of times (e.g., 20) and on each pass use a different seed for the random number generator. We compute the mean of the metric under study (e.g., CPD or CPI) across the 20 runs along with the sample standard deviation. We then vary a simulation parameter (like session size) to study scaling behavior and plot the 20-run mean (with the standard deviation indicated by an error bar) for each value of the parameter.

In our study, we found that CPD's are more or less bounded by feedback delays in the system. That is, it takes a receiver at least the amount of time in the feedback delay to react to congestion. The CPI on the other hand reflects the system's proximity to equilibrium. When the CPI is small, the network is in a transient phase in which many receivers are running join-experiments (or the network is potentially in an unstable or oscillatory state). When the CPI is large, congestion periods are infrequent and the system must therefore be in or close to equilibrium. However, if we simply averaged together the congestion periods intervals to compute the CPI, we would bias the result toward the transient-phase dynamics since there are many more congestion periods at that time. We therefore compute a time-weighted average of the CPI. That is, we compute the expected value of the instantaneous CPI for any given point in time. If congestion periods are given by

$$(t_1, t_2), (t_3, t_4), \ldots, (t_{N-1}, t_N)$$

then the time-averaged CPI is

$$\text{CPI} = (t_3 - t_2)\frac{t_3 - t_1}{t_N} + \ldots + (t_N - t_{N-1})\frac{t_N - t_{N-2}}{t_N}$$

This performance characterization is based only on congestion period statistics and therefore alone is not a comprehensive metric of overall system performance. For instance, congestion periods could be virtually eliminated by operating the system well below capacity and in this case aggregate throughput would be poor.

To address this shortcoming, we further characterize performance based on each receiver's proximity to its optimal throughput (i.e., optimal throughput is attained when the number of layers subscribed to is the maximum that the network can deliver without packet loss). In all of the single-source simulations, each receiver eventually reaches the optimal level of subscription. Above this optimum, the network is congested, and below, the network is underutilized. Except for infrequent and brief excursions due to join-experiments, each receiver maintains this level. Accordingly, the throughput can be made arbitrarily close to optimal by running the simulation arbitrarily long. Thus we evaluate throughput based on the time it takes the system to converge to the optimal operating point. We deem this time interval the *convergence time*. When combined with high CPI and low CPD, a good bound convergence time implies a well-functioning system. (We ignore the performance loss incurred by a mismatch between the discrete set of possible rates and the exact available bandwidth. In our simulations such mismatch is arbitrary but in practice is difficult to avoid.)

5.4.2 Experiments

We have simulated RLM in many topologies and configurations and in this section we present a number of simulations that explore the scalability of RLM in simple environments. Each of our simulations characterizes either *intra-session* or *inter-session* behavior. In the intra-session simulations, a single source sends to one or more receivers. There is no interaction among multiple sessions; we simply explore the dynamics of the protocol in isolation. We then extend our experiments to the inter-session case, where multiple sources send to multiple receivers. In these environments, we explore the interaction among multiple independent instances of RLM.

5.4.3 Results: Intra-session Behavior

In this section, we present the results of simulations on the the three topologies illustrated in Figure 5.6. These intra-session simulations all utilize one source sending to one or more receivers and thus exercise only those protocols dynamics related to a single session in a fixed and isolated environment.

Topology (1) consists of a single source and a single receiver separated by a bottleneck link. By analyzing the performance as we vary the latency on the bottleneck link, we explore the protocol's delay scalability.

Topology (2) extends topology (1) with multiple receivers. Here, we explore the scalability of the algorithm with respect to session size. As the size increases, we expect the join-experiment frequency during transients to increase and impact the packet loss rate. Likewise, in large sessions, join-experiments inevitably interfere with each other causing misinterpretation of the optimal capacity.

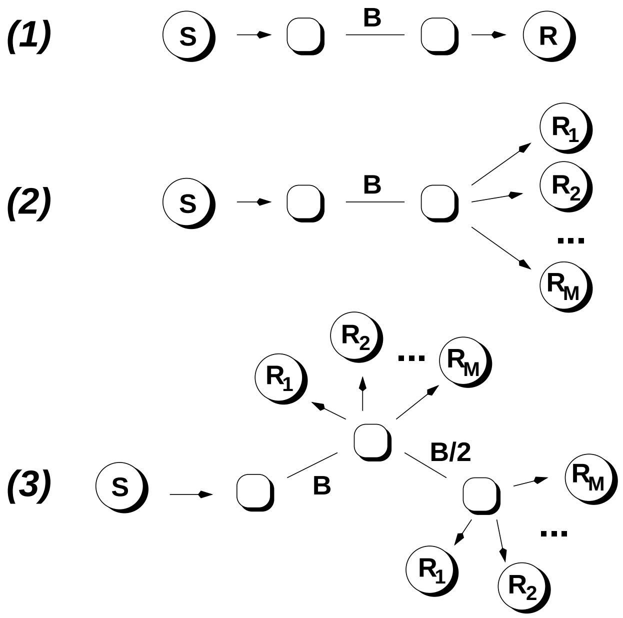


Figure 5.6: **Intra-session Topologies.** Our intra-session simulations explore the behavior of a single source sending to: (1) a single receiver, (2) multiple receivers, and (3) multiple receivers spread across two heterogeneous bandwidth clusters.

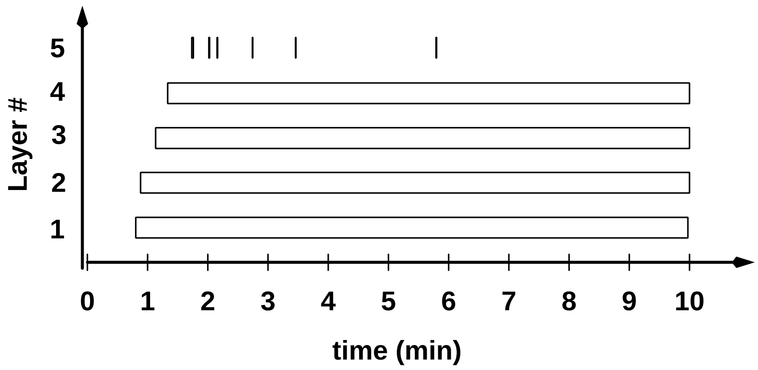


Figure 5.7: **RLM Sample Path.** A simple simulated sample path of topology (1) agrees with our conceptual model presented earlier.

Topology (3) explores the performance in the presence of bandwidth heterogeneity by considering two sets of receivers. The first set is connected at rate B while the second set is connected at rate $B/2$. In this scenario, the receivers downstream of the lower speed link must be robust against the high-bandwidth join-experiments from the other set of receivers.

Latency Scalability. In the first experiment, we placed a hierarchical CBR source at S in topology (1), ran RLM at node R, and fixed the link delay at 10 ms. The simulation was run for 10 (simulated) minutes. In this case, the behavior is predictable. The receiver ramps up to the number of layers supported by the link, then conducts join-experiments at progressively larger intervals until the maximum interval is reached. The duration of the join-experiment is roughly twice the link latency plus the queue build-up time; the impact of packet loss is proportional to the duration of the join-experiment, and thus proportional to the link latency.

This behavior is confirmed in Figure 5.7, which shows the level of subscription as it evolves over time for this simulation. Note that the receiver reaches the optimal layer subscription in about half a minute and at that point conducts join-experiments at progressively larger time intervals. Each join-experiment lasts less than a second.

To explore the delay sensitivity, we varied the link delay in topology (1) from 1 ms to 10 seconds and computed the mean CPD at each latency across 20 simulation runs. Figure 5.8 plots this average CPD vs. link latency (with the standard deviation indicated by the error bars). As the latency increases, we expect the impact from congestion to increase since it takes longer for the receiver to learn when loss occurs, prolonging congestion periods. At low latency, the overall delay is dominated by packet transmission time (i.e., the bandwidth); hence the reaction time is independent of the delay, which is reflected in the left half of the graph. But at higher link latency, propagation time dominates. Thus the protocol's reaction time linearly tracks the underlying link latency. Since the CPD is directly related to the reaction time, it is in turn linearly proportional to the underlying latency as indicated in the right half of the graph. In other words, the protocol behaves as expected — it does not become unstable even at high latencies and reacts as quickly as is feasible to congestion.

Session Scalability. In the next experiment, we varied the session size as illustrated in topology (2). Again, we fixed the link delays to 10 ms and ran each simulation for 10 minutes. Figure 5.9 shows the results. We plotted the time-weighted average of the congestion period interval (CPI) vs. session size. Because this configuration has multiple receivers, we computed the CPI from observations at just a single

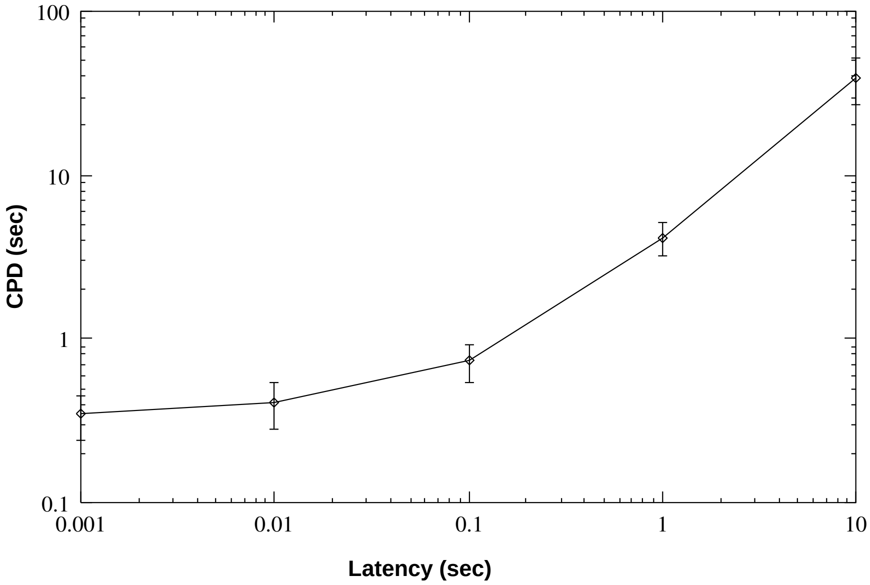


Figure 5.8: **Latency Scalability.** The sensitivity of congestion period duration (CPD) to propagation latency of a one-hop link. At high latency the end-to-end delay is dominated by propagation time and CPD tracks latency linearly.

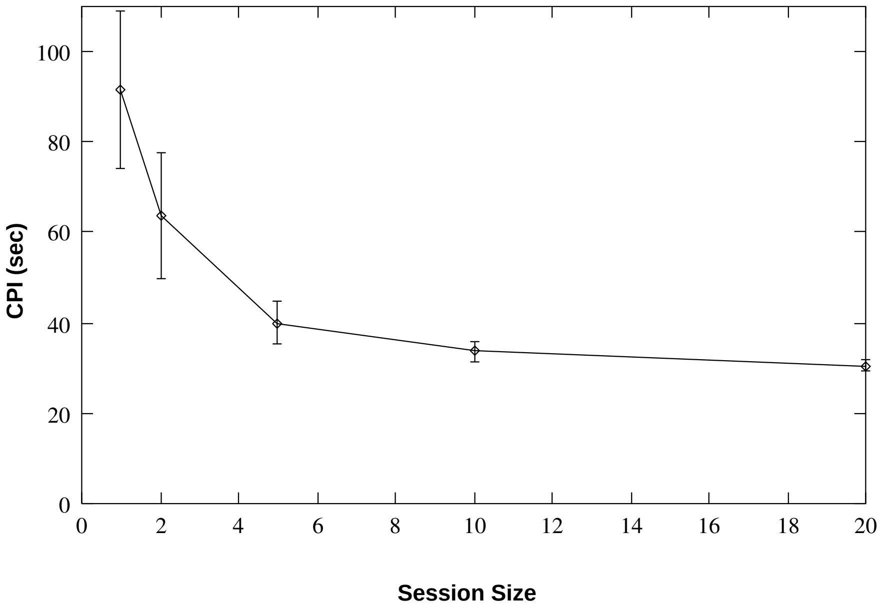


Figure 5.9: **Session Size Scalability.** The sensitivity of congestion period interval (CPI) on session sizes approaches the lower bound of the maximum join-timer random variable.

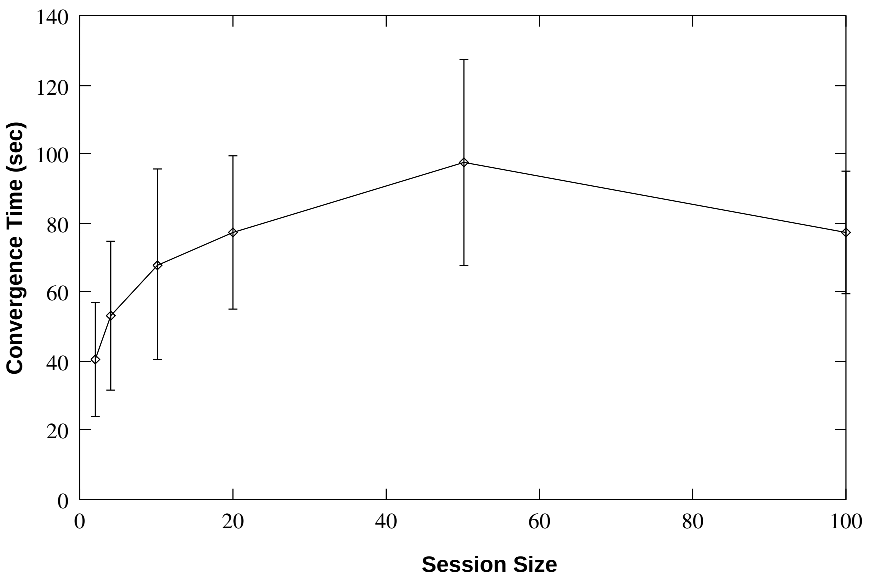


Figure 5.10: **Rate of Convergence.** The convergence time is well behaved as the session size grows. In fact, larger session sizes tend to stabilize the algorithm.

receiver (all receivers have equivalent statistics by symmetry). The graph shows that the CPI decreases (and hence the frequency of congestion periods increases) with session size. But the curve does not fall below 30 seconds, indicating a limit on the overall frequency of congestion periods. This 30 second bound is directly related to the maximum join-timer interval. Since this maximum is set to 60 seconds and the join-timer is chosen over a distribution bounded below by half this amount (i.e., 30 seconds), as more and more receivers become active, the likelihood that one of them chooses a value close to the lower bound increases. Moreover, because each receiver resets its join-timer upon observing a failed experiment, the group as a whole runs at the bounded rate.

In this second experiment we also explored how the session size of topology (2) impacts the rate of convergence of each receiver to its optimal level of subscription. Figure 5.10 plots the mean convergence time vs. session size averaged over 20 simulation runs. Each original data point represents the time it took a receiver to reach and maintain its optimal level of subscription (aside from infrequent join-experiments). While the convergence times grow with session size initially, past some point, they tend to stabilize with sufficiently large groups. On the one hand we expect the convergence time to increase with group size since experiments across the group interfere and suppress each other. But beyond some threshold, the group learning effect dominates and the convergence time is dominated by the group as a whole adapting each of their join-timers for the highest layer. Note that the standard deviation of each sample is relatively large, which results from the high degree of randomization applied to the join-timers.

Bandwidth Heterogeneity. Figure 5.11 shows that the algorithm works well even in the presence of large sets of receivers with different bandwidth constraints. Here we plot the CPI vs. session size for a configuration of topology (3), where we have a source sending to receivers that are partitioned into separate clusters with different reception bandwidth capabilities. From the graph we observe that the high bandwidth cluster (situated close to the source) experiences an increased CPI (i.e., a lower rate of congestion period)

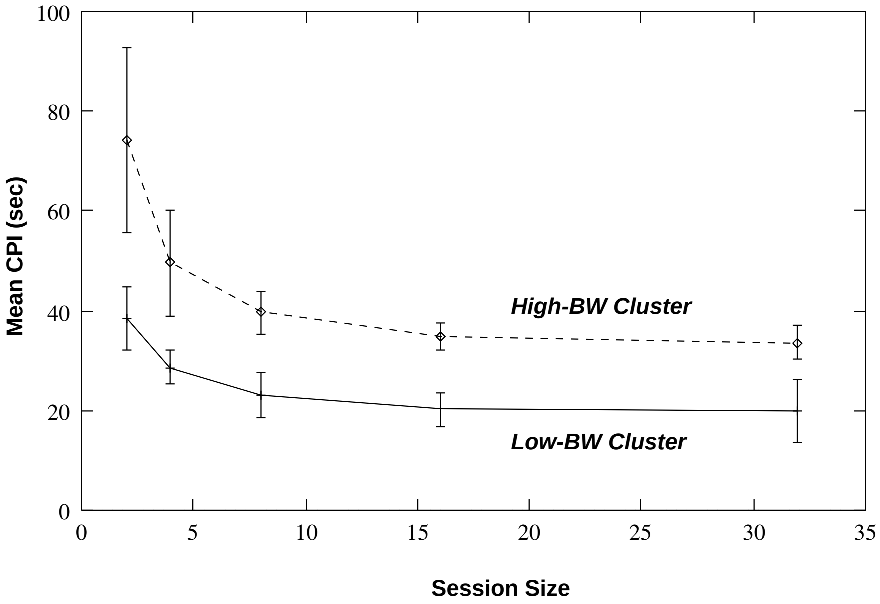


Figure 5.11: **Bandwidth Heterogeneity.** Even in an environment with heterogeneous bandwidth (i.e., two clusters behind different bottlenecks), the congestion period frequency is well behaved as we increase the group size.

than the low bandwidth cluster. This is because the low-bandwidth receivers see congestion periods that result from both the first link bottleneck as well as the second one (labeled $B/2$ in the topology). After the high-bandwidth receivers adapt to the first bottleneck and the low-bandwidth receivers adapt to the second bottleneck, join-experiments occur on each link with roughly the same frequency. Thus, receivers downstream of the second link see roughly double the rate of join-experiments relative to the upstream receivers. While this phenomenon affects the frequency of congestion events, the overall degree of impact is limited by the number of stable layers (which we assume to be a small constant). An area of future work is to elaborate the shared learning algorithm to take these disparities into account and thereby balance the join-experiment intervals across the layer hierarchy.

In this simulation we also measured the congestion period duration to ensure that bandwidth heterogeneity does not cause receivers to mis-infer that their experiment is causing congestion and fail to back off. Figure 5.12 plots the mean congestion period duration across 20 simulation runs at each session size. Although the congestion periods seen by the low-bandwidth cluster are longer than those for the high-bandwidth cluster, they are well behaved across the entire range of session sizes. Note that the increase in congestion period duration experienced by the low-bandwidth cluster is a direct consequence of the increased end-to-end latency that results from the additional network hop between the source and these receivers.

5.4.4 Results: Inter-session Behavior

In this section, we present the results of simulations for topologies that utilize multiple sources sending to multiple receivers and thus explore *intra-session* behavior, i.e., how the protocol dynamics evolve across multiple independent but mutually interacting sessions.

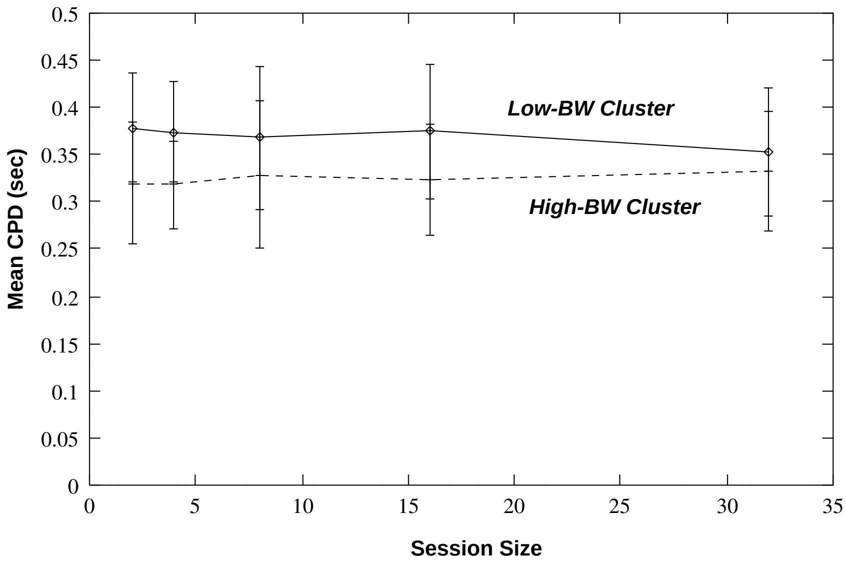


Figure 5.12: **Congestion Period Duration.** As with the congestion period frequency, the congestion period duration is well behaved in the face of heterogeneous bandwidth.

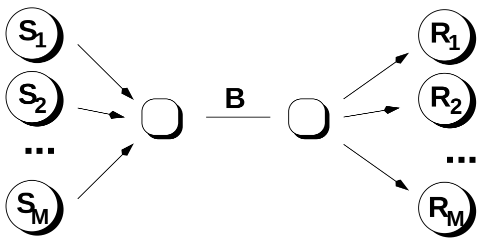


Figure 5.13: **Superposition Topology.** To study inter-session behavior, we superimpose several RLM sessions over a single link.

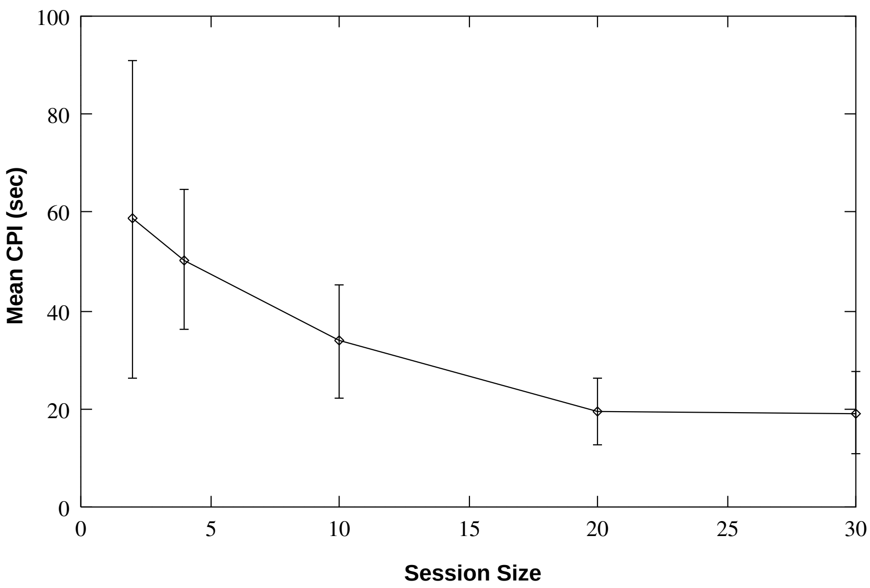


Figure 5.14: **Superposition.** RLM is well behaved under superposition of independent sources. Even though we do not carry out shared learning across independent sessions, congestion from one receiver's join-experiment suppresses other receiver's from initiating a new join-experiment.

Superposition. The topology shown in Figure 5.13 explores the performance of RLM when some number of independent single-source/single-receiver sessions share a common link. We ran several simulations and varied the number of source/receiver pairs. The bottleneck link bandwidth was scaled in proportion to the number of pairs and the router queue limit scaled to twice the bandwidth-delay product. Figure 5.14 illustrates the average congestion period interval as a function of the number of independent sessions. Even though we do not exercise the shared learning algorithm across independent sessions, the protocol is well behaved as the number of sessions increases. That is, as the session gets large the aggregate join-experiment rate does not increase without bound. Rather, congestion caused by competing but independent receivers tends to prevent other receivers from initiating new experiments.

Bandwidth Apportionment. While the previous simulation demonstrates that multiple independent RLM sessions interact gracefully, it says nothing about how "fair" that interaction is, i.e., how bandwidth is apportioned across the different sessions. If one session obtained all the bandwidth and thus starved all of the other sessions, the expected performance for any individual would be very poor.

To assess the fairness of bandwidth apportionment, we repeated the previous simulation a number of times for a fixed number of sessions (20) and analyzed the final level of subscription that each receiver obtained. Figure 5.15 plots a histogram that reflects the probability density for each receiver converging to the corresponding number of layers. For this configuration, the fair share for each receiver is ideally four layers. Although guaranteeing fair apportionment algorithmically on a strictly end-to-end basis is not possible, our simulation shows that (in this scenario) the probability of attaining four layers is reassuringly high — about 50%. Moreover, most of the mass of the density (92%) is concentrated in layers three through five, where a receiver diverges from optimal by at most one layer. Note that in all of these simulation runs, no receiver

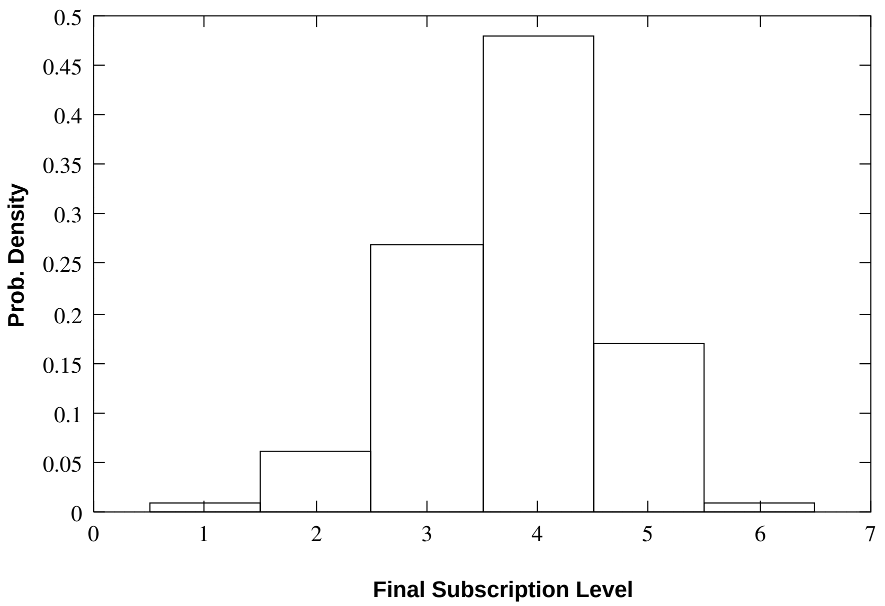


Figure 5.15: **Bandwidth Apportionment.** This graph plots a histogram of the subscription level after convergence of multiple, independent, single-source/single-receiver sessions. Even though RLM cannot guarantee fair-share apportionment of bandwidth across sessions, the system in this configuration tends toward a fair allocation.

was ever entirely starved of bandwidth.

Large-scale Network Behavior. The simulations presented thus far are relative small-scale with restricted heterogeneity deliberately for the purpose of isolating one variable under study. Larger simulations are more realistic but more difficult to interpret. Here we present one large-scale simulation to show that RLM has the potential to perform well in somewhat more realistic environments.

Figure 5.16 depicts a heterogeneous "network backbone" composed of network links that vary in bandwidth by a factor of 4. In these simulations, we place sources and receivers randomly at the edges of the backbone. Adjacent to each edge router is a cluster of end nodes that represent the leaf networks. In this diagram, we show each leaf as a three-node cluster, but the actual size varies on each simulation because of the randomized placement.

We ran a number of simulations over this topology with the highest bandwidth path set to 2040 kb/s. In each run, we placed 4 sources randomly in the network and for each source, placed a variable number of receivers (M) at random locations uniformly distributed across all edge routers (excluding the router adjacent to the source). We then varied M to explore RLM's performance under scaling of the receiver-set size. Figure 5.17 plots the CPI vs. the receiver-set size. The mean congestion period interval is roughly consistent across all of the receiver-set sizes and as expected tracks the maximum join-timer parameter. Relative to Figures 5.9 and 5.11, we do however note increased variability in this metric, notably for the case where $M = 20$. This is due to the interaction between independent sessions, which lack the determinacy provided by the shared learning algorithm.

Although these simulations show that the resulting congestion periods are limited as expected, we have not shown that the RLM streams each obtain good throughput. An area of future work is to compare

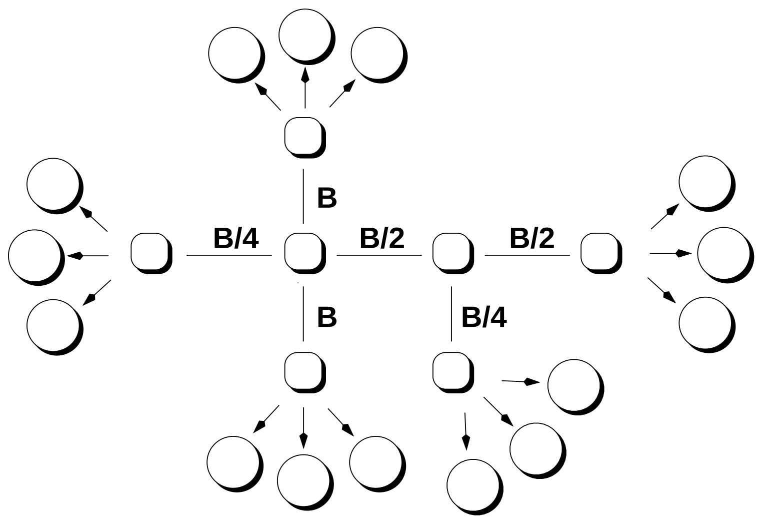


Figure 5.16: **Random Placement Topology.** To study larger scale more heterogeneous interactions, we place source and receivers randomly over this simulated "backbone" topology.

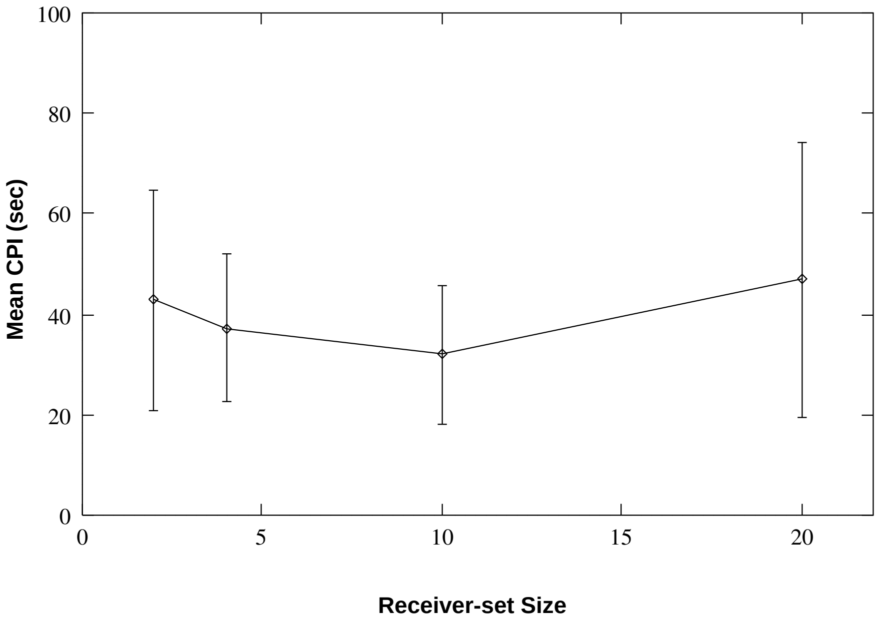


Figure 5.17: **Random Placement Simulation.** The mean congestion period interval is roughly fixed for increasing receiver-set sizes though its variability increases.

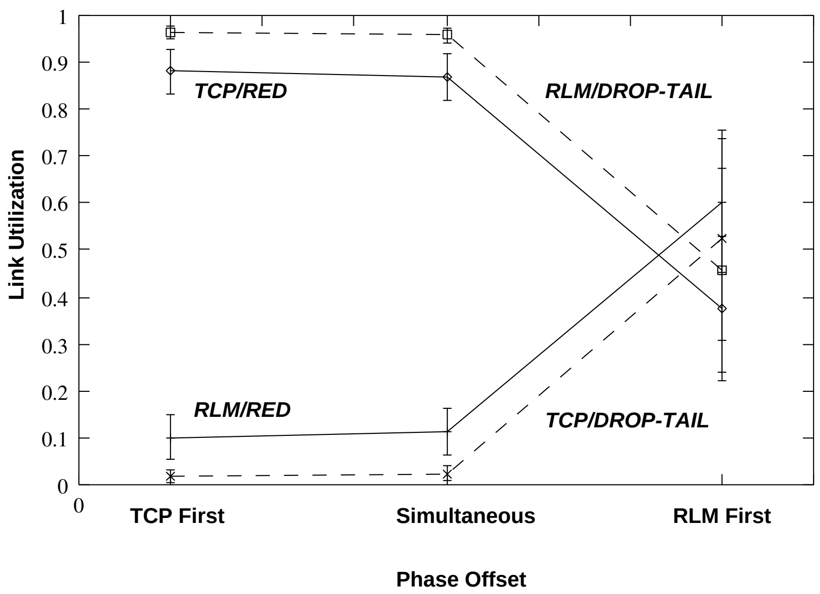


Figure 5.18: **RLM/TCP Interaction.** The interaction between RLM and TCP though stable can be unfair. If the RLM conversation starts and converges first, then the final bandwidth apportionment is approximately fair; otherwise, it is grossly unfair.

the results of this random placement simulation with those obtained by computing the optimal allocation of layers as described in [118].

5.4.5 Interaction with TCP

Because of the non-stationary and non-linear nature of RLM and other congestion control algorithms like TCP slow-start, formal analysis of their interaction is a hard, open problem and we thus cannot analytically predict their interdependent behavior. But to shed some light on their interaction, we conducted simulations of a single RLM flow and a single TCP conversation sharing a common link. We looked at three cases distinguished by the phase offsets between the two conversations, i.e., the order in which the transfers are initiated. In one scenario, RLM runs first, converges to equilibrium, and then TCP runs; in the second, they begin simultaneously; and in the third, TCP runs first. Since the interaction of the end-to-end dynamics of these two protocols depends on how packets are dropped in the network, we performed simulations with both standard drop-tail gateways as well as RED gateways.

We ran twenty simulation runs of this simple configuration for each of the three phase offsets. Figure 5.18 shows the results. The solid line represents simulations run under RED while the dashed line corresponds to drop-tail. In each case, we plot the utilization of the underlying link attained by each conversation (we computed the utilization only over the time period during which both connections were active). Each vertical slice contains two sets of 20 runs with error bars to represent the sample deviation. Under the RED scenario, when TCP starts first, RLM obtains a utilization of roughly 10% while TCP obtains about 90%. Under drop-tail, the disparity is even worse: the split is about 96%/4%. But if the RLM connection begins first, the bandwidth is about equally apportioned across both conversations under either scenario. In general,

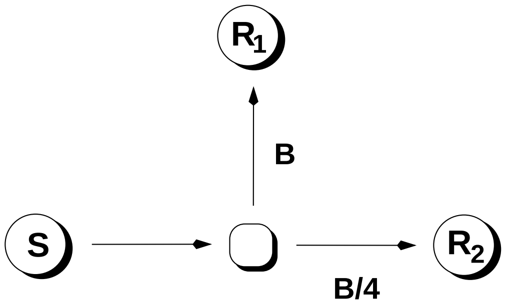


Figure 5.19: **RLM/IVS Test Topology.** A simple topology to explore the performance of RLM relative to IVS for two heterogeneous receivers.

RLM adaptation is less aggressive than TCP since it reacts to a single packet loss during its probe phase and hence has difficulty competing with an established TCP connection. In contrast, its hysteresis is comparable to that of TCP and thus the interaction is better behaved when RLM starts first.

Although not quantitatively presented here, the dynamics of both conversations are well-behaved and throughput was good in all three cases. RLM converges to a steady state while TCP converges to its normal congestion avoidance mode: it opens the window by one packet on each round-trip time, eventually drives the network into congestion, detects packet loss, shrinks the window in half, and repeats the process indefinitely.

One benefit of RED gateways is their ability to penalize flows in proportion to their bandwidth use and thereby encourage fair behavior among cooperative congestion control strategies. When TCP probes for capacity by exponentially growing its congestion window or when RLM probes for capacity by conducting a high-bandwidth join-experiment, a temporary excess of packets floods the bottleneck router. Consequently, RED is more likely to drop packets from this bandwidth-probing conversation. We see from Figure 5.18 that while RED helps, it is far from ideal. In some cases (i.e., when RLM starts first), bandwidth is apportioned reasonably well, while in others, it is apportioned poorly but at least under RED, no conversation experiences outright starvation. Under drop-tail, however, many simulation runs included large time spans where the receiver dropped to zero layers and was effectively starved of bandwidth.

5.4.6 Comparison with IVS Congestion Control

All of the previous simulations assess the dynamics of RLM in isolation, interacting with multiple instances of itself, or interacting with TCP. We now consider the performance of RLM's receiver-based adaptation relative to source-based control. To do so, we implemented a version of the *ivs* congestion control scheme in *ns* based both on the description in [15] and on a reference implementation. We separately simulated an IVS conversation[4] and an RLM conversation on the topology depicted in Figure 5.19. In both cases, we configured a source at S to become active at the beginning of the simulation simultaneously with a receiver labeled R_1. At time 300, receiver R_2 joins the session, resulting in a simple heterogeneous bandwidth demand.

Figure 5.20 shows the results of the two (separate) simulations composited into one plot, which

[4]For this discussion, we refer to the congestion control scheme in the *ivs* application as IVS.

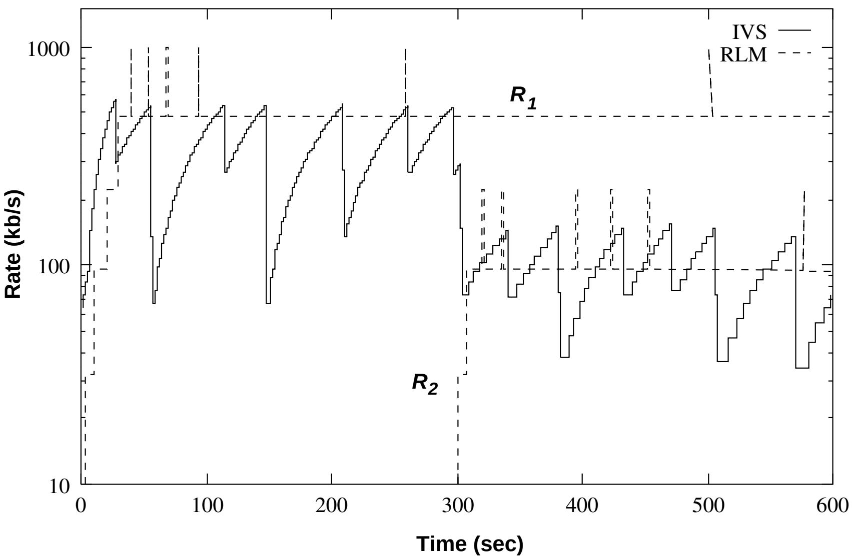


Figure 5.20: **Simple IVS/RLM Simulation.** Separate simulations of RLM and IVS superimposed here on one plot show the advantage of receiver- over sender-driven rate control. At time 300, a second receiver joins the session. IVS drops the single rate to accommodate the new receiver, but RLM splits the rate using layered transmission.

graphs the log of the bit-rate delivered to each receiver vs. time. The solid line corresponds to the IVS source/receiver, while the upper dashed line represents the RLM receiver at R_1 and the lower dashed line represents the RLM receiver at R_2. There is only one line for IVS because a single bit-rate is fixed at the source and both receivers see the same rate (modulo packet loss).

IVS uses a source-based congestion control strategy that starts with some initial rate, increases the rate linearly with time, and upon detecting congestion, multiplicatively decreases its transmission rate. As long as the source detects no congestion, it increases its transmission rate by a constant factor every few round-trip times. Once it detects congestion (via a scalable, probabilistic receiver-feedback algorithm), it cuts back its rate by a factor of two and continues its linear growth pattern. As a result, the IVS bit rate oscillates in a sawtooth fashion between the full bottleneck rate and half that amount. No damping is introduced into the control law, thus preventing the system from settling to a stable operating point. We note that this same ringing effect is apparent from actual measurements carried out in [15, Figure 5].

When the second receiver arrives at time 300, congestion occurs because of limited capacity along the S/R_2 path. R_2 eventually notifies the source, which in response throttles back its transmission rate on each round-trip time until the new receiver no longer reports congestion. Unfortunately, node R_1 must then endure the signal at 1/4 the original rate. Moreover, because IVS reacts conservatively (i.e., if only about 1% of the receivers report congestion), even if there were 50 or so receivers co-located with R_1, they would all receive reduced-rate video to avoid overloading the constrained link to R_2.

RLM on the other handles this condition gracefully[5]. As seen in the figure, the two receivers split

[5] The IVS congestion control scheme was a pioneering work in rate-adaptive video and its authors readily recognized the short-coming of source-based adaptation. In [15], layered transmission was briefly cited as a potential solution for accommodating band-width heterogeneity but the authors proposed no specific solution and suggested that such a system would require explicit support from network routers.

their rate and adapt to the capacity of the individual links. Moreover, the learning algorithm in RLM dampens the controller activity over time. Instead of continual oscillation as with IVS, the RLM receivers quickly learn appropriate levels of subscription and the discrete nature of the layers provides hysteresis that locks the rate at a constant level.

A final item to compare is the speed with which each algorithm finds the bottleneck rate. In this configuration, both schemes discover the bottleneck rate after roughly the same amount of time. But in general, if the disparity between the initial and bottleneck rates is greater and/or if the the initial IVS rate parameter is smaller, the exponential probe algorithm of RLM would discover the bottleneck rate before the linear increase algorithm of IVS.

5.5 Summary

In this chapter we proposed a protocol and distributed adaptation algorithm for the transmission of layered signals over heterogeneous networks using receiver-driven adaptation. We described our shared learning mechanism that gracefully accommodates large-scale sessions by distributing the learning process across the entire group. Through simulation, we evaluated the performance of RLM and showed that it incurs reasonable congestion overhead and convergence rates under several scaling scenarios both for intra-session as well as inter-session dynamics. Finally, we demonstrated the advantage of receiver- over sender-driven adaptation with a comparison of the RLM and IVS congestion control schemes.

Chapter 6

Low-complexity Video Coding for Receiver-driven Layered Multicast

Although the Receiver-driven Layered Multicast adaptation framework in the previous chapter provides a mechanism for distributing multirate flows to heterogeneous sets of receivers, it does not define the specific semantics of the individual flows or provide specific methods for adapting media representations for the layered architecture. To this end, we developed a video coding algorithm designed specifically for an environment like RLM, where we require a layered representation, good resilience to packet loss, and reasonably low complexity. As we argued in Chapter 4, we cannot solve this problem simply by taking a monolithic, "off the shelf" compression algorithm tailored for an altogether different environment. Instead we propose a new codec that embodies a novel blend of existing compression techniques. Although we exploit well-known techniques, we have brought them together in an original configuration, taking a systems-oriented approach that optimizes the algorithm across the network and into the end-system. By incorporating the application level framing (ALF) protocol architecture into the design of our codec, we create a deliberately tight coupling between the application and network that leads to good performance both in terms of high compression gain and in terms of low run-time complexity. In both these dimensions, our layered codec performs as well as or better than state of the art Internet video codecs.

In this chapter, we present the detailed design and the underlying algorithms of our layered video compression algorithm based on hybrid DCT/wavelet transform coding and hierarchical conditional replenishment. In the next section, we present our functional requirements and motivate the design outlined in subsequent sections. We then present our techniques for compressing the video sequence both temporally and spatially. Next we describe our optimizations that lead to a fast and efficient implementation and back up these claims with actual performance measurements. Finally, we present the packetization protocol that frames the PVH bit stream into packets for transmission across our layered multicast transmission channel.

6.1 Requirements

To complement RLM, our compression algorithm must satisfy a number of requirements:

- First, the bit stream must have a *layered representation* in order to interact with the RLM layered delivery model.

- Second, the algorithm must be *low-complexity*. Because we want to study the scaling behavior of our video delivery system, we must be able to deploy it on a large scale. One way to do this is to implement the codec in software, publicly distribute it, and have many people use it. In order to provide incentive for people to use it, the software must work well over a large range of machine capabilities and therefore must have an efficient implementation.
- Finally, because RLM drives the network into momentary periods of congestion and because the Internet environment is best-effort, loosely controlled, sometimes unpredictable, and involves bursty packet loss [14], the algorithm must have high *loss resilience*. That is when packets are dropped, the decoder should not have to wait long before re-synchronizing and the resulting errors should not persist unreasonably long or make the partially decoded video signal incomprehensible.

If an existing compression algorithm met all of these requirements, then we could simply incorporate it into our system. Unfortunately, no scheme currently does. For example, the ITU's H.261 and H.263 and ISO's MPEG-1 international standards do not provide layered representations and are all relatively sensitive to packet loss. Although the MPEG-2 standard does support layered representations, it does not operate efficiently at low bit rates because it relies on intra-frame updates, or I-Frames, to resynchronize the decoder in the presence of errors or packet loss. In order to make the decoder robust to loss, the I-Frame interval must be made relatively small, forcing the encoder to produce full frame updates relatively often. In many conference-style video sequences, there are large static backgrounds, and frequent I-Frame updates result in a highly redundant and inefficient transmission. Moreover, existing compression standards that were designed for hardware implementation over bit-oriented constant-rate channels impose undesirable constraints on software-based implementations for packet-switched networks. For example, an H.320 codec must compute an error-correcting polynomial and interleave bits from audio and video on non-byte boundaries — both trivial in hardware but cumbersome and inefficient in software.

Instead of a standardized compression algorithm, we could potentially adopt an existing experimental layered compression algorithm in our system. Taubman and Zakhor's 3D Subband Coding system is a high performance scalable video compression algorithm that produces a very fine-grained layered representation [162]. Its computational complexity, however, is relatively high and acceptable run-time performance will require a few more generations of processor evolution. Vishwanath and Chou's Weighted Wavelet Hierarchical Vector Quantization algorithm [173] is low-complexity and has a layered output format. Their algorithm is based entirely on table look-ups and runs fast on current generation hardware. However, they have not produced a publicly available implementation nor presented details on its overall performance in real environments. Although a table-driven approach may yield speed-ups on today's hardware, the ever-increasing performance gap between the processor and memory system may make such an approach less attractive in the future.

Given that no current algorithm satisfied all of our design constraints, we designed a new layered compression scheme based on our experiences adapting H.261 for Internet transmission [120]. To meet our goal of low-complexity, the algorithm is relatively simple and admits an efficient software implementation. Moreover, the software-based approach provides an easy route for incrementally improving the algorithm as technology improves and as we better understand how to achieve robust compression in the presence of packet loss.

In the following sections, we present our video compression algorithm by decomposing it into the two subproblems of temporal compression and spatial compression. Temporal compression attempts to reduce the bit rate by exploiting statistical correlations from frame to frame in an image sequence, while spatial compression attempts to eliminate redundancies by exploiting statistical correlations within a given frame.

Our algorithm employs a very simple model for temporal compression known as block-based conditional replenishment [120, 127], and uses a hybrid DCT/subband transform coding scheme for spatial compression. In the next section, we describe the conditional replenishment algorithm and in the subsequent section, we describe the spatial compression algorithm.

6.2 Temporal Compression

In Chapter 4, we argued that block-based conditional replenishment has a number of advantages over more traditional motion-compensating temporal prediction models and we therefore adopt it in our layered video compression algorithm. We now describe the detailed algorithmic steps of conditional replenishment: block selection, block aging, and temporal layering. Our scheme is derived in part from the conditional replenishment algorithm used by the Xerox PARC Network Video tool, *nv* [62].

6.2.1 Block Selection

To decide whether or not to encode and transmit a block, the conditional replenishment algorithm computes a distance between the reference block and the current block. As is standard practice with common motion-compensation algorithms, we run conditional replenishment exclusively off the luminance component of the video. The particular metric we use is an absolute sum of pixel luminance differences. If the block of reference pixels is $(r_1, r_2, \ldots, r_n)$, the block of new pixels is $(x_1, x_2, \ldots, x_n)$, and the threshold is T, then the new block is selected if

$$\left|\sum_{k=1}^{n}(r_k - x_k)\right| > T$$

We use an absolute sum of differences rather than a sum of absolute differences for several reasons. First, because the background noise process is zero-mean, a sum of differences tends to filter out the noise while a sum of absolute differences amplifies it. Hence, the threshold becomes more sensitive to the noise level. Second, since motion artifacts tend to have a strong DC bias, the sum of differences will successfully extract this bias. Finally, the sum of differences is less expensive to compute (i.e., it uses one rather than many absolute value operations).

Unfortunately, changes to a small portion of a block are not detected by our distance metric alone because it is hard to disambiguate noise and isolated changes without sophisticated analysis. We solve this problem by exploiting the fact that frame-to-frame changes typically result from scene motion or camera pan, and both of these processes create large spans of spatially correlated pixels. Hence, we assume that isolated changes occur to a block only when there are large changes to an adjacent block. We give up on detecting small, isolated changes and simply "spread" the block selection decision from one block to adjacent blocks. While we have found that this algorithm works well most of the time, certain types of image sequences cause problems (e.g., small mouse cursors on a video-captured display or a laser pointer on a captured projection screen).

The exact choice of the threshold T is not particularly critical. We found heuristically that values ranging from 40 to 80 or so all work reasonably well across different camera types and lighting conditions. Our current implementation uses a fixed value of 48. We conjecture that the metric might be improved by accounting for the average luminance value of the input, but have not yet experimented with this approach or any other methods of adaptation because the current algorithm works well enough in practice.

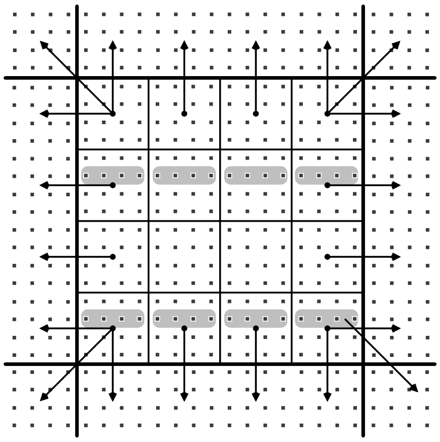


Figure 6.1: **Block Selection Algorithm.** Block selection is carried out on a 4x4 grid (thin lines) that determines if the containing 16x16 block (thick lines) is replenished. As indicated by the arrows, updates are spread to adjacent 16x16 blocks to minimize "small motion" artifacts.

Figure 6.1 illustrates the basic block selection and spreading algorithm. Unlike *nv*, which uses a "flat" algorithm that operates on 8x8 blocks, we use a two-tiered algorithm that carries out selection and spreading over a 4x4 grid, which in turn, is used to update 16x16 blocks. The diagram shows each pixel as a small square dot, the 4x4 cells as thin lines, and the 16x16 block as thick lines. If any of the cells that comprise a block are selected, then that entire 16x16 block is encoded. Furthermore, each selected cell is spread to adjacent blocks as indicated by the arrows in the diagram. For example, if the lower left cell is selected, then the three adjacent blocks (at 180, 225, and 270 degrees) are also selected. The four internal cells cause no spreading.

As a performance optimization, we do not process all 256 pixels of a block (or all 16 pixels of a cell) on each new frame. Instead, we compute the selection decision using only one row of pixels per cell across two rows of cells per block. (If we process just one row of cells, artifacts would arise from the fact that spreading only reaches 8 pixels but the block is 16 pixels high.) The figure illustrates an example subset of pixels, highlighted in gray, that are inspected by the algorithm on some arbitrary frame. There are 8 sets of 4-pixel groups. We compute the distance to the reference pixels individually on each 4-pixel group and if the difference is above the threshold, we send the containing block and possibly an adjacent block (according to the cell spreading rule).

By using only a subset of the rows, the memory traffic for the selection process is significantly reduced (e.g., by a factor of 8 as described in §6.5). Moreover, the algorithm interacts well with a typical memory subsystem where processor cache lines are aligned with each row of pixels. To ensure that each line of the block is eventually examined, we inspect a different row on each new frame. A vertical row offset is maintained for each 8 horizontal lines, and after each new frame, we advance the offset by 3. We add 3 rather than 1 to increase the likelihood that we encounter an isolated change faster. Since 3 and 8 are co-prime, the offset cyclically precesses through all eight lines.

A 16x16 replenishment unit has two advantages over an 8x8 unit. First, the overhead in representing the block addressing information is reduced since there are four-fold fewer blocks and therefore four-fold fewer location codes. Second, the larger block size in general allows more efficient spatial compression

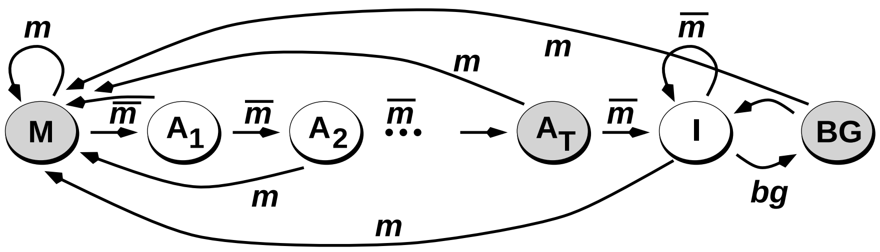


Figure 6.2: **Block Aging Algorithm.** A separate finite-state machine is maintained for each block in the image. State transitions are based on the presence (m) or absence ($\overline{m}$) of motion within the block. A background fill process spontaneously promotes a small number of idle blocks to the background state (bg). The block is replenished in the shaded states.

techniques [93]. However, larger blocks cause coarse grained image updates, which consequently results in redundant transmission, limiting the potential improvement of an arbitrarily large block.

6.2.2 Robust Refresh

The threshold in the block selection algorithm provides hysteresis by suppressing block updates when there is little change. Unfortunately, this hysteresis causes minor but noticeable blocking artifacts. The problem can be explained as follows. Consider a block that is static, changes due to motion, then returns to a static state. In effect, the block travels along a trajectory from its initial state to its final state. At some point before its final state, the block selection hysteresis takes hold and the block is no longer replenished even though the block continues to change. Hence, the final block has a persistent error with respect to the final static state.

We can solve this problem with a refresh heuristic. When the selection algorithm ceases to send a given block, we age the block and re-send it at some later time. Presumably, by then, the block will have reached its final state along the "change trajectory" and the refresh will counteract the artifact.

We carry out this "robust refresh" algorithm using the finite-state machine (FSM) illustrated in Figure 6.2. Each block in the image has a separate FSM and we encode and transmit a block only in the shaded states. Whenever the block selection algorithm detects motion in a block, the state machine transitions to the motion state (labeled M). When there is no motion, the FSM transitions through a number of aging states. At the age threshold (state A_T), we send the block, and in turn, enter the idle state (I). In the current implementation, we fix A_T at 31. At high frame rates, this translates into approximately one second of delay, which is sufficient time for motion artifacts to decay. At low frame rates, the lag is longer because A_T does not depend on the frame rate and hence causes a more persistent artifact.

We additionally run a background fill process to continuously refresh all the blocks in the image to guarantee that lost blocks are eventually retransmitted and that the entire image is filled in for receivers that join an in-progress transmission. This process selects some number of idle blocks in each frame and spontaneously transitions them to the background state (BG).

By supplying the FSM state information for each block to the encoder, adaptive quantization can be utilized to substantially improve the perceived quality of the reconstructed video. Since block updates at the age threshold are less frequent than those in the motion state and since the aged block is likely to persist

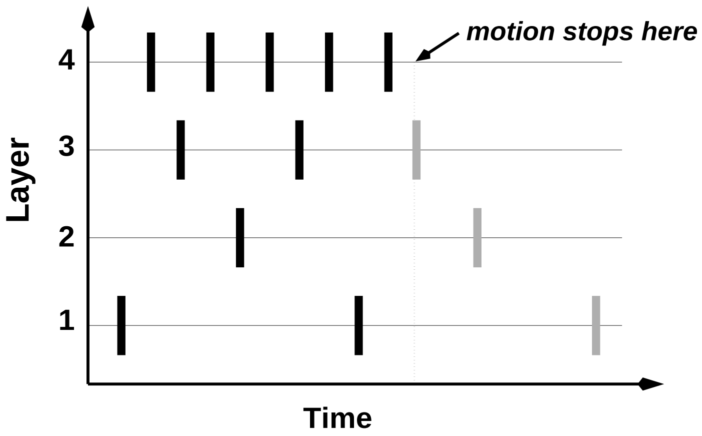


Figure 6.3: **Temporal Layering.** We extend the conditional replenishment algorithm to produce multiple rates by striping block updates across different output layers. When a block becomes idle, we "slide it" down the layer hierarchy to guarantee that the most up-to-date version appears on the base layer.

into the future, it is advantageous to spend extra bits to code such blocks at a higher quality. Similarly, because background blocks are sent infrequently, we can send them at the highest quality with little increase in overall rate, causing static scenes (like screen captures of projected slides) to eventually attain high fidelity. Upon implementing this scheme in an early version of *vic*, the utility of the tool for video-captured viewgraph transmission increased substantially.

6.2.3 Temporal Layering

The conditional replenishment algorithm described above generates a single rate of block updates for a given input frame rate. We can extend the algorithm to produce multiple rates in a temporal hierarchy by splitting block updates into separate layers. One well-known approach for creating a temporal hierarchy is temporal subband decomposition. To this end, we could carry out subband analysis on a block granularity and extend the block update across the next power of two interval for which the block remains active. Unfortunately, this introduces complexity and extra delay over simple conditional replenishment.

Instead, we utilize our robust block refresh algorithm and stripe block updates across different layers to provide multiple frame rates. To produce a graceful degradation in frame rates, we arrange the subsampled frames so that any set of layers produces frames spaced evenly over time. We do this as follows. Assuming there are $M + 1$ layers, we assign layer $L_M(n)$ to all block updates during frame time n, where

$$L_M(n) = M - r(n \bmod 2^M + 2^M) + 1$$

with

$$r(n) = \min\{k > 0 : \lfloor n/2^k \rfloor 2^k \neq n\} - 1$$

i.e., $r(n)$ is the bit position (numbered from 0) of the right-most non-zero bit in the binary representation of n.

The hierarchy that results in the case for $M = 4$ is shown in Figure 6.3. If the receiver processes all four layers, then the resulting frame rate is maximal. If the receiver processes only three layers, the frame rate is half the maximum rate. For two layers, it is one-fourth, and so on.

As long as a block is continuously transmitted, this scheme works well. But when a block undergoing motion becomes inactive and its last update occurs on any layer k with $k > 1$, that block position will be inconsistent on all layers l such that $l < k$. A simple remedy is to force the block update in the age-threshold state onto layer 1, thereby limiting the time extent of the inconsistency. We tried this approach, but the qualitative performance was unsatisfactory because the block artifacts were too noticeable for too long. Instead, when a block becomes inactive at time n_0, we transmit it additionally at times given by

$$\min\{n \geq n_0 : L_M(n) = k\}$$

for $k = 1 \ldots L_M(n_0)$. In other words, after a block becomes inactive, it "slides down" the layer hierarchy. As indicated by the gray blocks in Figure 6.3, we transmit a block update at each inferior layer down to layer 1. At that point, the block undergoes the aging algorithm and is eventually re-sent on layer 1 in the age-threshold state.

6.2.4 Temporal Layering Overhead: Measurements and Analysis

The overhead incurred by the redundant block transmissions is not as great as it may seem. Because the redundant block updates only occur after a block under motion becomes inactive, the overall redundancy is inversely proportional the length of this "active period". Moreover, the redundancy present in lower-rate layers, where bandwidth is critical, is less than that in higher-rate layers. For example, layer 1 alone never has a redundant block update, while the full hierarchy contains the maximum number of redundant updates.

To assess the impact of the redundant block updates, we carried out some simple analysis and measurements. First, we derive an expression for the expected number of redundant block updates with respect to each layer as a function of the layer's position in the hierarchy. Assume there are N layers numbered 1 through N. Since the frame rate doubles at each layer, the probability that a block becomes inactive is 1/2 on layer N, 1/4 on layer N-1, 1/8 on layer N-2, and so forth (assuming that active period termination points are uniformly distributed across time). The number of redundant block updates with respect to layer m is $m - 1$. Hence, the expected number of redundant block updates for layer m is

$$R(m) = \sum_{k=2}^{m} \frac{k-1}{2^{m-k+1}}$$

Table 6.1 gives the value of $R(m)$ for $m = 2, 3$, and 4. For large m

$$R(m) \approx m - 2$$

so the redundancy is linearly proportional to the number of block updates. Fortunately, there is a practical limit on the number of temporal layers. Since the frame rates grow exponentially, just four temporal layers is sufficient to provide a range from 30 down to 3.75 f/s. Thus the overhead is bounded.

Given an expression for the redundancy of block updates, we can compute a bound for the expected overhead for a given distribution of active periods. To this end, we measured the block update process of a typical MBone transmission (the UCB Multimedia and Graphics seminar). We collected statistics over approximately one hour of video and reduced the data to the sample distribution plotted in Figure 6.4. Using this

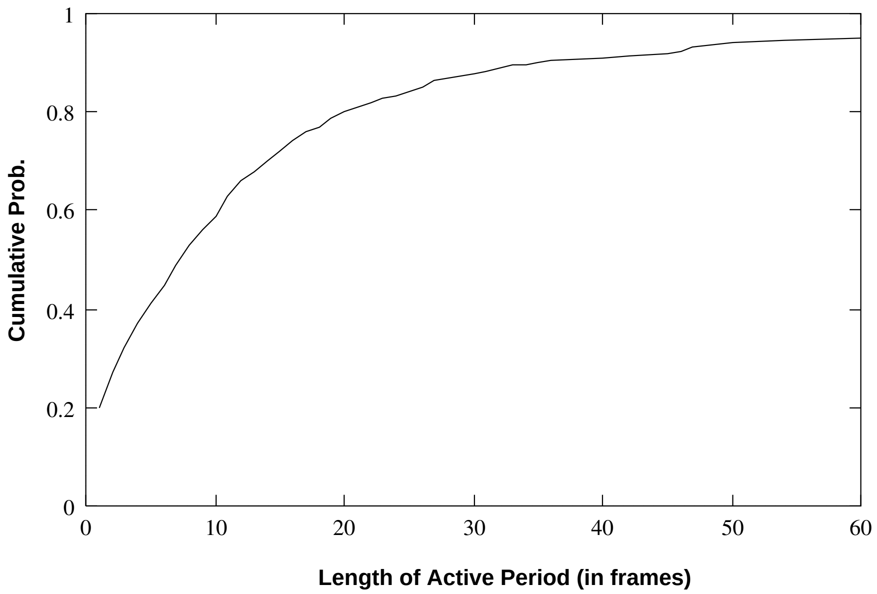


Figure 6.4: **Probability Distribution of Active Periods.** To assess the impact of redundant block updates, we measured the probability distribution of a block's "active period". This graph plots a sample distribution of the random variable that represents the duration of the active period seen by each block update (i.e., longer active periods are weighted by the fact that they contain more block updates).

m	$R(m)$	$\hat{R}(m)$	$R^*(m)$
2	0.5	0.39	0.16
3	1.25	0.98	0.28
4	2.125	1.7	0.31

Table 6.1: Overhead of Redundant Block Updates

distribution, we can compute the expected number of redundant updates for each transmitted block, $\hat{R}(m)$, from the following expression:

$$\begin{aligned}\hat{R}(m) &\leq \sum_{1}^{\infty} P(\text{block falls in interval of length } k) \times \text{per-block overhead} \\ &= \sum_{1}^{\infty} P(X = k)\frac{R(m)}{k}\end{aligned}$$

where X is the random variable that represents the distribution in Figure 6.4. We note that this value is an upper bound since we conservatively assume that active periods are spaced sufficiently far apart so that the maximum number of redundant blocks are transmitted. That is, we overestimate the amount of redundancy when the next active period begins before the last redundant block is sent (i.e., because the block update is no longer redundant but we count it as such). Table 6.1 gives the expected overhead for 2, 3, and 4 temporal layers using the sample distribution collected from the simulation.

To obtain a more accurate measure of the overhead, we additionally simulated the hierarchical block replenishment algorithm using the raw block-update trace as input. We computed the ratio of redundant blocks to the total number sent, $R^*(m)$. These results, listed in the fourth column of Table 6.1, are more promising. For a two-layer temporal hierarchy, the overhead is only 15% and for 3 and 4 layers, it is around 30%. Note that the gap between $m = 3$ and 4 is small relative to that between $m = 2$ and 3, which is a consequence of the fact that the four-layer hierarchy induces much more overlap between adjacent active intervals.

Overall, the overhead due to redundant block transmissions, while non-negligible, is reasonably low for layers 1 and 2, where the overhead is more likely to have an impact (i.e., at low bit rate). We believe this performance compromise is a reasonable design tradeoff since we gain the robustness of conditional replenishment without sacrificing a temporal hierarchy.

6.3 Spatial Compression

After the conditional replenishment stage selects blocks for transmission, they are compressed spatially. In this section, we describe the layered spatial compression algorithm that is applied to each block.

The first version of our coder [122] utilized subband decomposition since this approach induces an inherently layered representation. In this coder, we carry out subband decomposition over the entire image and then use pixel-domain conditional replenishment to determine the subband coefficients to transmit. We first perform subband analysis horizontally across the image to yield low- and high-frequency representations of the signal, commonly called the L and H subbands. In turn, we apply the same low/high frequency decomposition vertically yielding a total of four subbands: the coarse-scale LL subband, containing a low resolution version of the signal, and the enhancement subbands containing horizontal detail (HL), vertical detail (LH) and diagonal detail (HH). After subband analysis, we encode those subband coefficients whose basis vectors are spatially centered over each selected pixel block. We then group the coefficients across scales with like orientation into the well-known quad-tree structure, and then entropy-code them using a variant of Shapiro's scheme for Embedded Zerotrees of Wavelet coefficients (EZW) [157]. This coding structure is illustrated in Figure 6.5.

Unfortunately, a tension arises between subband decomposition and conditional replenishment. While subband decomposition induces a multiscale structure where transform coefficients correspond to multiple overlapping regions of the image, conditional replenishment assumes spatially confined pixel blocks.

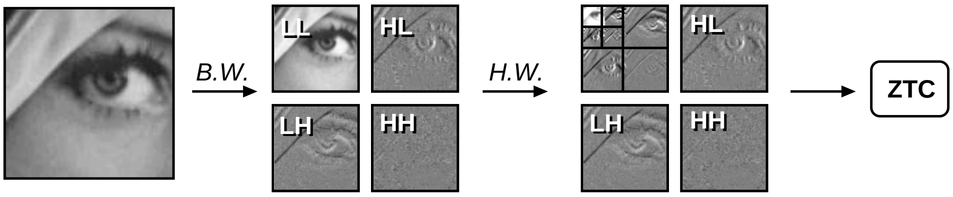


Figure 6.5: **Zerotree Wavelet Coding Structure.** We decompose a pixel block using our 1/3/3/1 4-tap biorthogonal wavelet (*B.W.*), and in turn, transform the LL subband with a Haar wavelet (*H.W*). The resulting subband coefficient hierarchy is entropy-coded using zerotrees (ZTC).

Moreover, in traditional subband coding systems the analysis/synthesis filters are relatively long and, when iterated, generate basis vectors that span large regions of the image. While this has attractive properties for multiresolution representation (i.e., one can achieve very good low-resolution approximations at low bit rate), it is a poor match to the block replenishment model. Our solution for the coder described above was to use short analysis filters to increase the coherence between the subband and pixel representations. We used the following biorthogonal filters for the first-stage analysis [169]:

$$\begin{aligned} H_0(z) &= -1 + 3z^{-1} + 3z^{-2} - z^{-3} \\ H_1(z) &= -1 + 3z^{-1} - 3z^{-2} + z^{-3} \end{aligned}$$

with the following synthesis[1]

$$\begin{aligned} G_0(z) &= (1 + 3z^{-1} + 3z^{-2} + z^{-3})/16 \\ G_1(z) &= (-1 - 3z^{-1} + 3z^{-2} + z^{-3})/16 \end{aligned}$$

and Haar filters for the remaining three stages. Because a four-tap filter induces only one pixel of overlap, and because the Haar basis vectors induce no additional overlap, we can exploit pixel-domain conditional replenishment to determine which subband coefficients to encode.

Although this codec outperforms several existing Internet video coding schemes, its compression performance is somewhat inferior to the commonly used Intra-H.261 format [120]. To carry out ongoing, large-scale experiments within the MBone user community, we rely on active use of the applications, protocols, and compression formats. Our experience is that a few isolated experiments do not provide the level of feedback necessary to evolve a robust and thoroughly tuned codec design that interacts gracefully with the network. To encourage the largest possible user community to participate in experiments with the new format, we felt that it was necessary to produce a layered codec that outperforms the best existing practice.

6.3.1 PVH: A Hybrid Transform

Our approach for improving the compression performance of our wavelet coder is to leverage off the compression advantages of the Discrete Cosine Transform (DCT) for block-oriented processing. In the wavelet coder described above, the first stage of subband decomposition generates an 8x8 block of coarse-scale subband coefficients. Since this coarse-scale block represents a low-resolution version of the original

[1] Note that we use the more *regular* filters at synthesis, where regularity implies that the iterated filter bank converges to a smooth basis.

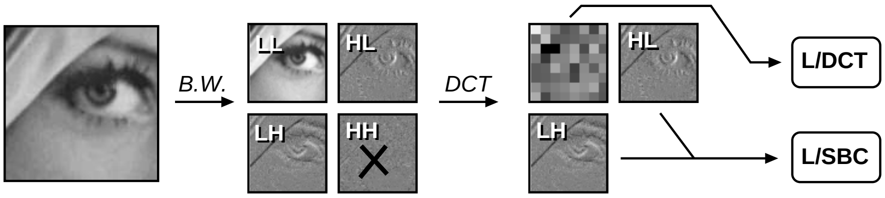


Figure 6.6: **Hybrid Transform Coding Structure.** We decompose a pixel block using our 1/3/3/1 4-tap biorthogonal wavelet (*B.W.*), and in turn, transform the LL subband with a DCT. The resulting DCT coefficients are run-length/entropy coded and progressively refined (L/DCT = "layered DCT"). The LH/HL subband coefficients are progressively coded by compressing them a bit-plane at a time using a quad-tree decomposition (L/SBC = "layered subband coefficients").

image, its statistics are consistent with a typical image signal. Hence, a coding scheme tailored for normal images will work well on the coarse-scale LL subband [109]. Rather than carry out additional subband decomposition using the Haar transform on the LL subband, we instead apply an 8x8 DCT as depicted in Figure 6.6.

To retain an embedded bit stream, we encode the transform coefficients progressively by coding the DCT coefficients a bit-plane at a time. Our technique is similar to the point transform used in progressive-mode JPEG [85, Annex G] and the SNR-scalability profile in MPEG-2. We code the DCT coefficients in a number of passes. In the first pass, the DC coefficient is quantized and coded (using spatial DPCM across blocks), while the AC coefficients are quantized to a power of 2, scanned in "zig-zag" order, and run-length/entropy coded in a fashion similar to JPEG, MPEG, or H.261. This "base-layer" pass is followed by a number of enhancement passes, which are in turn, decomposed into a refinement pass and an identification pass. Each new pass corresponds to an additional bit of precision:

- **Refinement.** In the refinement pass, an additional bit of precision of the magnitude of each previously transmitted coefficient is sent verbatim (there is little opportunity to compress these refinement bits).

- **Identification.** In the identification pass, coefficients that become non-zero at the current quantization level are transmitted (along with their sign). These coefficients are identified simply by a series of run codes, interleaved with sign bits, and terminated by an end-of-block symbol. As in JPEG, the coefficient positions that have already been sent are skipped in the calculation of the run-codes. This decreases the entropy of the run-codes and therefore increases the compression efficiency.

By decomposing the compression process into a number of passes that successively refine the transform coefficients, we can easily format the bit stream into a layered representation. Although DCT-based coding of the LL coarse scale band has been previously proposed [109], as far as we know, the combination of progressive DCT transmission and multiresolution subband decomposition has not been explored.

Simultaneously with the progressive coding of DCT coefficients, we encode the LH and HL subband coefficients using a simple quad-tree decomposition of bit-planes. Unfortunately, we must sacrifice the compression advantages of zerotrees since we no longer carry out multiple levels of subband decomposition, and hence, cannot use zerotrees to predict information across scales. We experimented with a version of the algorithm that additionally applied a DCT to the 8x8 LH and HL bands but found that this provided negligible improvement.We discard the HH band altogether as it typically contributes little energy to the reconstructed signal.

Conceptually, the progressive coding of subband coefficients is carried out as follows. We represent the coefficients in sign/magnitude form and scan the coefficient bit-planes one plane at a time, from most significant bit to least significant bit. We code a bit-plane as follows:

- If size of bit-plane is one bit, output that bit.
- Otherwise:
  - If all bits are zero, output 0.
  - Otherwise, output 1. If this is the most significant bit of the magnitude of this position, output the sign. Divide bit-plane into four equally sized bit-planes, and recursively code these subplanes.

This decomposition is similar to the "Autoadaptive Block Coding" algorithm of Kunt and Johsen [106] though they applied it to bi-level images without any transformation. The *hcompress* algorithm described in [176] similarly exploits this technique in combination with subband decomposition over the entire image.

In practice, our algorithm diverges somewhat from this conceptual framework in order to optimize the syntax for better run-time performance. Instead of carrying out a separate pass for every bit-plane, the first several planes are grouped together and treated as a quantized coefficient. This reduces the run-time overhead since we process multiple layers in parallel as is done by the "Layered-DCT" implementation in [3]. We scan the subband coefficients in a quad-tree fashion as described above and entropy-code each non-zero coefficient that is identified in the scan. Although we could group refinement passes into multiple passes as well, this is unnecessary because each additional level of precision typically induces an exponential increase in bit rate (at least until a large fraction of the coefficients have been found to be significant). Consequently, the rate allocation algorithm rarely assigns more than one refinement pass to an output layer. When it does, we simply allocate multiple sequential passes to the same layer. Finally, the output codewords are rearranged to facilitate a performance optimization described later. Version 1 of this codec bit syntax, which we call Progressive Video with Hybrid transform (PVH), is detailed in the appendix.

6.3.2 Bit Allocation

To optimize the compression performance of PVH, we must partition the rate between the DCT and subband coding subprocesses in an intelligent fashion. For example, if we allocated all of the rate to the subband coefficients, then the resulting image would be a "ghost image" composed of fine-scale edges in a gray background. On the other hand, if we allocated all of the rate to the DCT coefficients, then we would code noise in the DCT transform coefficients without recovering any of the fine-scale details. Clearly, the optimal allocation is not at either of these extremes.

Figure 6.7 plots a family of operational distortion-rate curves generated by coding the 512 by 512 grayscale *Lena* image with our hybrid coder. Each separate curve corresponds to a fixed number of refinement passes over the subband coefficients, or conversely, to the amount of quantization applied to each subband. In turn, we swept out each individual curve by successively increasing the number of refinement passes applied to the DCT transform coefficients. The best combinations of quantizers occur along the upper convex hull of the family of curves, i.e., for a given rate constraint, the quality is maximal along this curve. Hence, we achieve the best performance by partitioning rate to each subprocess according to the convex hull.

One approach for choosing these quantizers is to run an on-line optimization that continually updates quantization mix to reflect the changing signal statistics. By including codes to adaptively adjust the

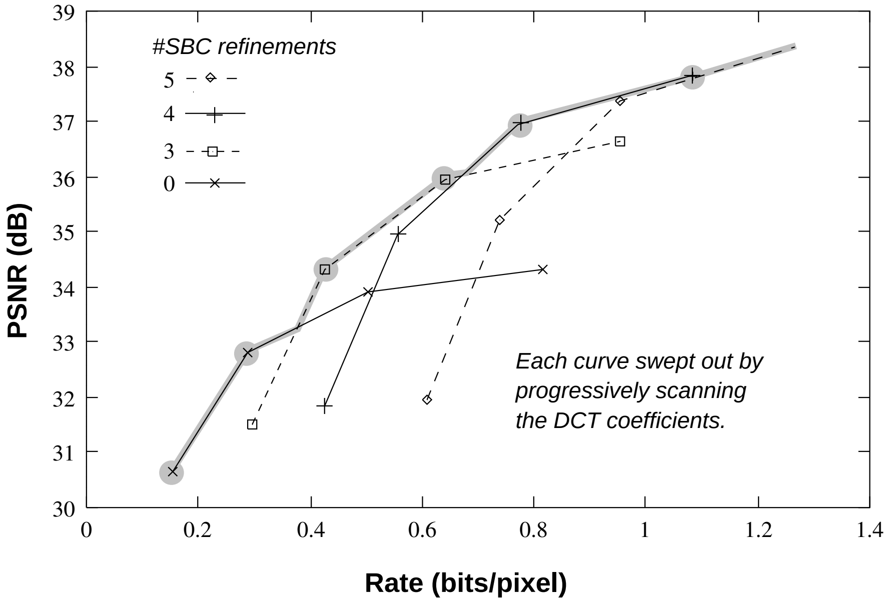


Figure 6.7: **Bit Allocation.** We determine the best mix of rate across the DCT and subband coefficients by computing the convex hull of a family of curves. Each curve is swept out by progressively scanning the DCT coefficients of the LL subband and each separate curve corresponds to a fixed set of LH/HL coefficient refinement passes.

quantization mix at the start of each block, we can perform adaptation on a block granularity. Since the sub-process distortions are additive (by linearity of the DCT and subband transforms), we could use a dynamic program to find a good approximation of the optimal solution [129].

Unfortunately, computing an on-line, adaptive optimization algorithm like this adds complexity that inhibits real-time performance. An alternative approach is to pre-select a fixed set of quantizers by hand and hope that they are never far from optimal. We do exactly this in our prototype because it is much simpler to implement and incurs no overhead. Using the Lena rate-distortion curves from above, we derive the progressive quantization structure given in Table 6.2. The *BL* columns indicate whether the corresponding base layer is present and the *REF* columns indicate the number of bits of refinement to the luminance DCT (LD), luminance subband (LS), or chrominance DCT (CD) coefficients[2]. The DCT chrominance refinements were chosen by hand based on visual inspection of quality and rate since our PSNR metric does not account for the color dimension. The luminance and chrominance DCT base-layer coefficients are quantized with a uniform quantizer of magnitude 32, while the SBC base-layer coefficients are quantized by 16. Note how the chrominance base layer is distributed on layer 1, resulting in a grayscale-to-color transition from layer 0 to layer 1. This overall decomposition gives a total of five spatial layers, which when convolved with the temporal hierarchy, produces a rich set of tunable output rates.

While this scheme has low-complexity and is simple to implement, the compression performance may be suboptimal if the input signal statistics do not match those of Lena. We tested the sensitivity of the optimal choice of quantizers to signal statistics by computing the optimum for several images from the USC

[2]There are no chrominance subband coefficients because the 16x16 chrominance planes are directly subsampled by 2 and each resulting 8x8 block is coded exclusively with the progressive DCT.

layer	*LD-BL*	*LD-REF*	*LS-BL*	*LS-REF*	*CD-BL*	*CD-REF*
0	X	0		0		0
1		1		0	X	0
2		0	X	0		0
3		1		0		1
4		0		1		1

Table 6.2: Layered Bit Allocation

image data base. In each case, the result was the same as that for Lena. Although optimal quantization selection is in general strongly image-dependent, our relatively constrained choice of quantizers limits their variability. Because our successive quantization scheme uses full powers of two, there are only a small number of refinement passes and the distance in distortion between quantizers is relatively large. Hence, there is little opportunity for the optimal points to shift.

On the other hand, the rate partitioning algorithm is carried out only at the encoder and can be later improved. Since we embed the quantization information into the output stream, our fixed, hand-chosen quantizers can be replaced with the automated process described earlier without changing the codec syntax.

6.3.3 Compression Performance

We compared PVH with two prevalent compression schemes for Internet video to assess its compression performance. These existing algorithms include the native format used by *nv* and the Intra-H.261 format used by *vic*. Because these schemes use similar conditional replenishment algorithms, we can compare their two-dimensional compression performance to assess their overall performance. Hence, we removed temporal coding overheads (like macroblock addressing codes) from each codec. Because we compare only grayscale PSNR performance, we additionally removed chrominance syntax overhead. In addition to the Internet video codecs, we compared our results against Shapiro's EZW algorithm [157] and progressive-mode JPEG [85, Annex G] to gauge the performance of our scheme against well-established subband- and DCT-based image codecs. For each algorithm, we obtained a distortion-rate characteristic for the 512x512 Lena gray scale test image as follows:

- **Intra-H.261.** We modified the Intra-H.261 coder from *vic* for arbitrarily-sized images and omitted macroblock addressing codes and chrominance processing. We obtained the rate-distortion curve by varying the standard H.261 quantizer.

- **NV.** We modified the *nv* coder for grayscale operation and omitted block addressing codes. We obtained the curve by varying the Haar coefficient dead zone.

- **PVH.** We used our prototype PVH coder with subband/DCT quantizers chosen by inspection according to Figure 6.7.

- **Progressive-JPEG.** We employed Release 6 of the Independent JPEG Group codec in grayscale and progressive modes. We obtained the curve using the JPEG codec's "scans" option to compute multiple operating points by controlling the number of refinement passes used by the encoder.

- **EZW.** We used the performance results reported in [157].

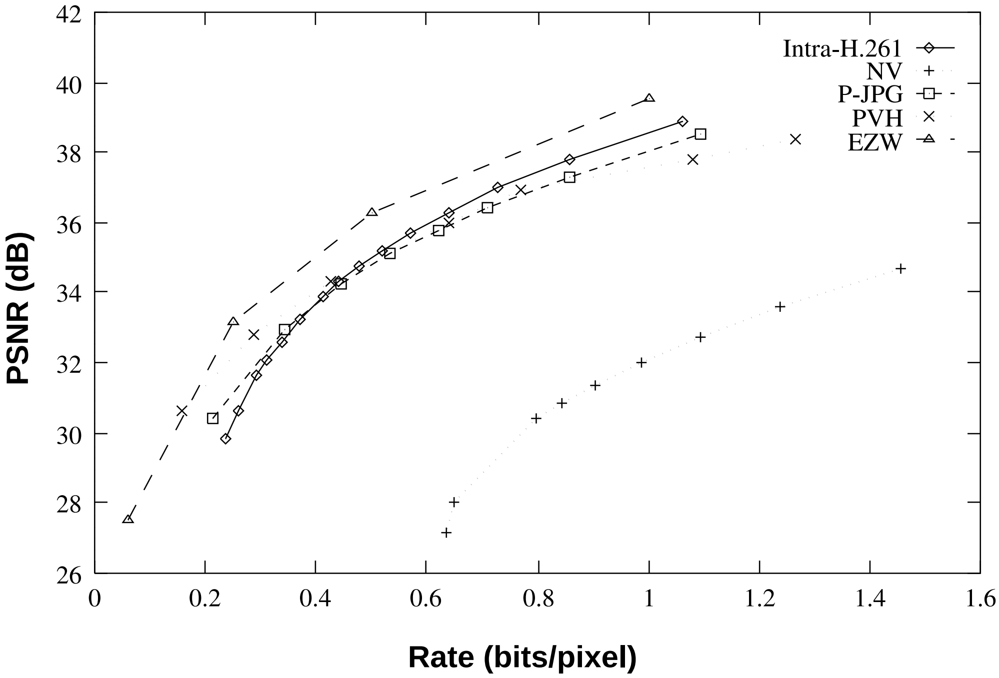


Figure 6.8: **Relative Compression Performance.** The compression performance of PVH is better than Intra-H.261 at low rates, comparable at medium rates, and somewhat inferior at high rates.

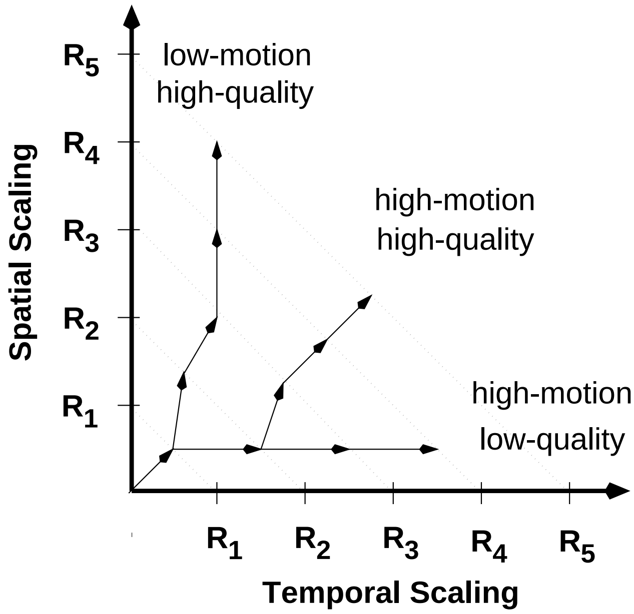


Figure 6.9: **Temporal/Spatial Scaling.** We cannot scale the spatial and temporal qualities simultaneously. Instead, we must choose a single path through this rate-scaling space. We show three such paths: The lower path leads to a high-motion/low-quality signal, the upper path leads to a low-motion/high-quality signal, and the middle path is a compromise between the two.

Figure 6.8 shows the results. Although EZW outperforms all of the other schemes, it has high complexity and cannot be used with conditional replenishment because its wavelet domain representation is not localized to blocks. At low rates, PVH performs as good as EZW and better than Progressive-JPEG. At roughly one bit/pixel and beyond, PVH performs 0.5 to 1dB below both Progressive JPEG and Intra-H.261. At these rates, PVH spends a significant fraction of its bit budget coding the fine-scale subband coefficients, which do not benefit from any lower-resolution conditioning information. The *nv* coding algorithm is about 6dB below the rest; for a fixed level of quality, the rate performance is two to four times worse. In summary, over the commonly used low-rate quality ranges, PVH outperforms existing Internet video formats and performs near or close to the other schemes at high rate.

6.4 The Spatio-temporal Hierarchy

Layered conditional replenishment and layered spatial compression together form a two-dimensional space over which we can scale the overall bit rate. But unfortunately, we cannot adjust both dimensions independently at each receiver — from the perspective of the network, the aggregate bit rate is just one parameter.

Figure 6.9 illustrates the tradeoff involved in scaling rate over the two-dimensional space. The vertical axis represents the rate allocated to improving spatial quality while the horizontal axis represents the rate allocated to improving temporal quality. A point in the upper left region corresponds to low frame rate and high spatial quality, while a point in the lower right corresponds to high frame rate and low spatial quality. The aggregate rate is the sum of the two coordinates. Hence, the isolines of fixed rate are straight lines with slope -1. When we increase the rate, say from rate R_2 to R_3, we can move from a point on the R_2 isoline to any point along the R_3 isoline that is reachable by a vector with direction 0 to 90 degrees. The problem then is to plot a *single* trajectory through this two-dimensional space to obtain a layered stream with a one-dimensional rate parameter. We call the trajectory through this two-dimensional space the *layering policy*.

The layering policy is a free parameter that should match the application context. For example, when the video channel is used to transmit seminar slides, spatial quality must be high so that the slides are readable. Likewise if the application is educational instruction of art history, then spatial quality should be high to faithfully represent illustrative artwork. On the other hand, if the speaker's slides are distributed over a separate "whiteboard channel", then many users would prefer high frame-rate at the cost of lower spatial quality to provide a heightened "sense of presence" of the remote location. Unfortunately, we must fix a single layering policy at the source and this prevents us from satisfying conflicting user desires.

We define a layering policy explicitly through the method by which temporal and spatial hierarchies are combined into a single layered stream. The problem is to map some number of spatial layers and temporal layers into some number of output or network layers. Ideally we would simply stripe mixtures of bits from the temporal and spatial layers across the appropriate output layers. However, this scheme works only if the temporal layers appear explicitly as bits to transmit. For example, in subband decomposition, temporal information is represented as explicit enhancement information to a coarse-scale temporal (i.e., blurred) signal. But in layered conditional replenishment, temporal layers do not appear as bits to transmit. Rather, the algorithm shifts spatial layers up and down the output layer hierarchy over time. For example, let $S_1 \dots S_N$ be a set of spatial layers and $L_1(n) \dots L_M(n)$ a set of output layers indexed by the frame number, n. Suppose we want two temporal layers and three output layers ($M = 3$). Then, the following assignment of spatial information to output layers gives the desired spatio-temporal structure:

$$\begin{array}{rcll} L_1(n) & = & S_1 & n \text{ even} \\ & = & \emptyset & n \text{ odd} \\ L_2(n) & = & \emptyset & n \text{ even} \\ & = & S_1 & n \text{ odd} \\ L_3(n) & = & S_2 & \end{array}$$

Layer 1 provides a low-rate low-quality signal, layer 2 doubles the frame rate, and layer 3 enhances the spatial quality.

A richer example is illustrated in Figure 6.10. Here we have three spatial layers and three temporal layers. Layer 1 alone provides the lowest quality, lowest frame-rate signal. Layer 2 increases the spatial quality but leaves the frame rate fixed. From there, layer 3 doubles the frame rate without changing the spatial quality. Layer 4 again doubles the frame rate. Finally, layer 5 refines the spatial quality to its maximum level. Note how we manipulate the frame rate for a given level of subscription by dynamically varying the output channel assigned to each spatial layer.

More generally, we define a map from spatial layers to output channels that varies over time according to the layered replenishment algorithm. In the previous two examples, the amount of spatial quantization is fixed for any subset of the layers but we can extend the scheme to dynamically adjust the allocation, for instance, to meet different bit rate constraints for each layer. We must solve an optimization problem that places constraints on the rate limit of each layer by scheduling the selection of quantizers and temporal hierarchy to smoothly adapt to changing input signal statistics.

For our particular codec, a general solution to this problem is still an open issue. We currently employ a simple interim strategy that works adequately in many contexts. In this approach, we control the

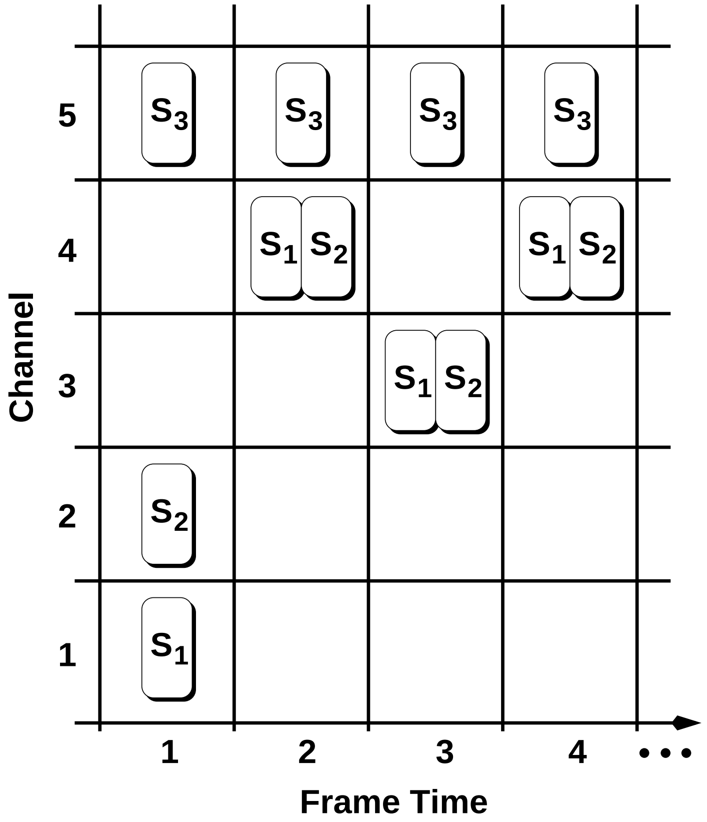


Figure 6.10: **Spatio-temporal Layering.** We combine layered conditional replenishment with the spatial compression algorithm to induce a spatio-temporal hierarchy where the allocation of spatial layers to network channels evolves over time.

bit rate of the base temporal layer, which may be composed of multiple spatial layers, by running it at a variable frame-rate to match the target rate. Whenever we transmit bits on this base layer, we schedule the subsequent frame time adequately far into the future to obey the rate limit. Accordingly, if the input video has high activity and motion, then the frame updates are large, the interframe time increases, and the frame rate drops. Conversely, if there is low activity, the frame rate increases. Since the frame times of successive temporal layers are tied to the base layer, we distribute the temporal hierarchy evenly over each frame-update interval.

Though far from perfect, we believe that this rate-control policy is reasonable in an environment like the MBone. Here we might want to limit the rate of a low-quality subset for the MBone, but distribute the remainder of the hierarchy locally without explicit rate limits. Additionally, we could decompose a 128 kb/s MBone layer into two spatial layers where the bottom most layer could be transmitted over narrowband ISDN.

Because the layout is completely configurable at the encoder, the layering policy can be freely manipulated without modification to the decoder. Accordingly, we can incrementally deploy improved versions of rate allocation algorithms without requiring global codec upgrades.

6.5 Run-time Performance

Now that we have described the basic compression algorithm, we turn to implementation issues and discuss the algorithm's complexity and how we achieve a fast implementation. First of all, we reduce run-time overhead compared to traditional DCT-based schemes though our use of subband decomposition. Instead of computing four relatively expensive DCT's and progressively coding all four blocks of DCT coefficients, we carry out one stage of subband analysis using inexpensive filters, code only one 8x8 block of DCT coefficients, code two 8x8 enhancement subbands with a fast algorithm, and discard the 8x8 HH subband. Although subband coding algorithms generally have higher complexity than DCT-based schemes, the combination of cheap filters and an inexpensive algorithm for encoding subband coefficients reduces the overall complexity.

We exploit a number of optimizations to speed up the encoding and decoding of DCT coefficients. At the encoder, we maintain the DCT coefficients in a sparse array. On the initial base-layer pass, we collect up the coefficients that are needed in later passes and store them in a temporary array. Since there are typically many zero-valued coefficients and we make multiple passes over the coefficients, the abbreviated array reduces loop overhead and memory traffic.

At the decoder, we store the DCT coefficients in the normal block-array format, but use a 64 element bit-vector to identify the significant coefficients (on a modern architecture, this bit-vector fits in a processor register). For each non-zero coefficient, the corresponding bit is set; otherwise, it is clear. This data structure improves performance in two ways:

- We avoid initializing the DCT coefficient array to zero on each new block. Instead, we simply clear the bit-vector.
- We carry out abbreviated processing of the refinement stages by structuring loops to skip over missing coefficients quickly using bit-wise logic that efficiently detects and skips over contiguous runs of zeros.

Conditional replenishment is the first stage of compression and requires access to only a subset of the pixels in a given block. If we decide to skip a block at this stage, we avoid all further processing. This approach complements video capture architectures that use Direct Memory Access (DMA) to transfer each digitized frame directly into memory, lifting the burden of processing uncompressed, high-rate video off the CPU. Since most of the pixels are (potentially) never referenced, much of the video data never needs to enter the CPU or processor cache. In our implementation, only 32 of the 256 pixels that make up a block are accessed, resulting in an eight-fold reduction in CPU/memory traffic.

We compute the subband coefficient quad-trees for each bit-plane in parallel with a single pass over the data. At the quad-tree leaves, we perform a bit-wise "OR" over 7-bit magnitudes of the four coefficients that comprise a leaf. For a 16x16 block, this gives eight trees each with seven bit planes, giving 56 binary-valued elements (again, this 56 element bit-vector fits in a 64-bit processor register). We then compute internal nodes of the quad-tree using bit-wise "OR" operations over the appropriate subsets of the 56 element bit-vector. In practice, not all bit-planes are needed and we collapse the first several planes into a single layer, allowing us to carry out these computations in 32-bits.

Additionally, we improve performance by using only shifts and adds to compute the subband analysis filter. Further, we can compute these operations in parallel using the parallelism inherent in a 32- or 64-bit ALU. Several new processor architectures provide 8-bit parallel add instructions to do exactly this (e.g., SPARC VIS, Intel MMX, and HP PA-RISC), but even on traditional architectures, we exploit ALU parallelism by inserting guards in the machine word. For example, to process a row of samples, we initialize a 64-bit register with 8 pixels (or coefficients) in a single memory load. We mask out every other pixel, perform several operations, then place the result back in memory with a single store instruction. Moreover, we check for overflow of several results simultaneously using a single conditional to reduce the number of branches in the inner-loop.

We optimize the Huffman decoding stage with a table-driven design. In this scheme, we buffer the head of the bit stream in a processor register and parse the next Huffman codeword with a table look-up. If the longest legal codeword is N bits, then we use the next N bits to index the table. The table entry provides the length L (with $L \leq N$) of the codeword and the corresponding symbol S. To decode the next symbol, we form an index from the next N bits in the bit-buffer, locate the table entry, discard L bits from the bitstream, and process S according to the codec syntax. We can additionally enhance memory locality, thereby improving processor cache performance, by using a two-tiered look-up table. Since the goal of a Huffman code is to minimize the average codeword size, the typical codeword length is small. Hence, we can construct an abbreviated table that contains the most frequently appearing codewords and is indexed by only M bits of input (with $M < N$). However, the codewords whose lengths are greater than M collide with other codewords in the table. In this case, the table entry contains an ESCAPE code that instructs the decoder to use a slower but completely defined operation (e.g., a full-sized table lookup). The Berkeley MPEG decoder [134] uses a similar table-driven approach.

Several operations are combined or are carried out "in-place" to reduce processor/memory traffic:

- The subband analysis stage performs quantization "on the fly" so that the output coefficients are stored in 8-bit format. This reduces memory traffic by a factor of 4 over full-precision representation.
- We place the output of the inverse DCT directly into the LL subband coefficient buffer.
- We combine the first stage of subband reconstruction, the conversion from sign-magnitude to two's-complement numerical form, and the the coefficient centering step (i.e., the step that biases each coefficient to the midrange of the quantization interval) all into a single pass.

We implemented PVH and these optimizations in our video conferencing application *vic* and compared its performance with the widely used Intra-H.261 codec [120]. As a simple quantitative assessment, we measured the run-time performance of both codecs within *vic* on an SGI Indy (200MHz MIPS R4400) using the built-in VINO video device. To measure the maximum sustainable compression rate, we disabled the bandwidth and frame rate controls for both coders and ran the test on an unloaded machine. We measured the resulting frame rates by decoding the streams on a separate machine. We configured the PVH coder with enough DCT and subband refinement layers to give quality roughly equivalent to that of the Intra-H.261 coder with its quantizer set to "5" (based on visual inspection and the Lena rate-distortion curves), and provided both coders with (approximately) the same, "high motion" 320x240 video input. The results were remarkably consistent across the two coders as they both generated output at approximately 11 frames per second. Because both schemes were limited only by the workstation's fixed computational resources, the run-time performance for this level of quality is roughly equivalent. For a typical "talking head" sequence with low scene activity, both encoders perform close to real-time (20-30 f/s).

6.6 Packetization

We have thus far described the RLM network protocol and the complementary PVH video codec that was co-designed with RLM, but the overall system is still incomplete because we have not specified the machinery to map PVH bit streams onto network packets for transmission across multiple communication layers. One approach for packetizing the PVH bit stream is to use a simple fragmentation protocol. Here a source simply breaks its bit stream into arbitrary packet-sized fragments and receivers reconstruct the original stream by reassembling these fragments. But this approach interacts poorly with the Internet protocol architecture because network packets can be lost, reordered, duplicated, or delayed. Under these conditions, we must be able to process packets from multiple, interdependent layers in an efficient and robust fashion.

To this end, we might attempt to build a modular, "black box" protocol that could provide generic semantics to cope with packet loss, delay, and reordering. However, such a protocol would poorly match our layered video stream. For example, the protocol could not know about specific relationships between the packets in different layers (without a complex programming interface), and thus would not know how to best proceed in the presence of loss. If a base-layer packet is lost, then all of the dependent packets may have to be discarded. On the other hand, if an enhancement layer packet is lost, then decoding can proceed, but only for some subset of the received packets. This is just one example of application semantics that cannot be easily expressed in a generic network protocol.

Instead we adopt Clark and Tennenhouse's ALF protocol architecture by developing a framing scheme that is carried out explicitly by and within the application. As we described in Chapter 4, ALF allows us to optimize the application explicitly for the network by reflecting an application's semantics in the design of its network protocol. Fortunately, a great deal of machinery for packet video transmission is already defined by the Real-time Transport Protocol or RTP [153] (see Chapter 3). Because RTP is based on ALF and meets many of our design goals, we leverage upon RTP in our design of a framing protocol for PVH.

Although RTP has proven to be a solid foundation for interoperable real-time audio/video applications, it was designed without any explicit notion of a layered signal representation. In RTP, each signal source is identified with a randomly allocated 32-bit source identifier (Source-ID) that is unique only within a single session (a collision detection algorithm resolves conflicts). Additionally, each user is identified with a variable-length "canonical name" (CNAME) string that is globally unique. Data packets are identified only by Source-ID, and each application periodically broadcasts a binding between its CNAME and Source-ID. Thus, a receiver can collate streams across different sessions (identified by different Source-ID's) using the level of indirection provided by the CNAME. With this framework, we can readily handle layered compression formats by treating each layer as a distinct "RTP session" and distributing it on its own multicast group.

However, the "RTP session per layer" approach adds unnecessary complexity. Not only does it force each receiver to manage all the CNAME/Source-ID bindings, but it requires newly arrived receivers to wait for the binding advertisement before they can start decoding a stream. Another problem is that it creates new error recovery conditions for dealing with conflicting information arriving on the different sessions.

A better approach is to extend the semantics of RTP. We proposed that a single Source-ID space be used across all the layers and that the allocation algorithm be carried out only on the base layer (which reaches all session members). Moreover, certain components of the RTCP control protocol like sender identification strings are redundant across the different layers and should only be sent on the base layer. These proposed changes allow a participant to use one Source-ID consistently across the logically distinct RTP sessions comprising the RLM/PVH hierarchy. This proposal is currently under review by the IETF[3] [160].

6.6.1 The PVH Framing Protocol

The flexibility of RTP's ALF-based framework gives us the freedom to optimize the PVH framing protocol for robust interaction with the underlying network. We based our framing protocol in part on our work adapting H.261 for resilient packet transmission in *vic* described in Chapter 4. A key property of the Intra-H.261 framing protocol is that packets are independent of each other and can be decoded in isolation or in arbitrary order (up to a frame boundary). This simplifies loss recovery since the start of each packet provides an explicit resynchronization point.

Ideally, we would like to incorporate the "idempotent" nature of Intra-H.261 packets into our PVH framing protocol, but unfortunately, this is not entirely possible with the layered approach. A fundamental problem is the necessary dependence between the packets at different layers within the spatial hierarchy. For example, block address codes appear only on the base layer. Thus, in order to decode enhancement layer packets, we must know the positioning context from the base layer. During decoding, we can propagate this conditioning information across the hierarchy by either processing packets in a carefully defined order and retaining information to provide later context or by grouping related packets and decoding the group as a unit.

At one extreme, we buffer, reassemble, and decode all of the packets of an entire frame. At the other extreme, we process each packet as it arrives, assuming all necessary earlier context arrives first. Within a frame, the decoder can process the spatial layers either sequentially or in parallel. In sequential decoding, all the blocks of a given layer are processed before advancing to the next layer, while in parallel decoding, all the layers of a given block are decoded before advancing to the next block. These different approaches involve implementation complexity and efficiency tradeoffs. For example, parallel decoding yields good memory-system locality (and hence good cache behavior) since each block is processed in its entirety before moving on.

We decided to develop a framing protocol that would provide enough flexibility to allow either the parallel or the sequential decoding method without incurring an unreasonable header overhead. Hence, we adopted a group-based framing protocol that allows the receiver to decode the bit stream in units smaller than a frame. To enhance loss recovery, groups are independent of each other — a packet loss in one group cannot adversely impact another group. Although groups are independent, a packet may straddle two groups. To account for this, PVH includes "resumption offsets" that indicate the offset into the packet at which the new group begins. Thus the decoder can process a subsequent group without first decoding the previous group.

[3] We jointly developed an Internet Draft describing extensions to RTP for layered media streams with Michael Speer of Sun Microsystems.

Slice-based Framing. Borrowing terminology from the MPEG specification, we define an idempotent decoding unit or *slice* as a range of coded image blocks. Each PVH packet header indicates the block addresses of the first and last blocks encoded in the packet, and we associate a slice with the block range of exactly one base-layer packet. That is, each base-layer packet induces a slice defined by that packet plus those packets at higher layers within the same frame (i.e., with the same RTP timestamp) whose block addresses overlap.

To identify and decode all the packets in this slice-oriented fashion, we must:

(1) identify each base-layer packet,

(2) indicate how spatial layers are mapped onto network channels, and

(3) specify how the encoded bit stream is allocated across the spatial hierarchy.

First, we must identify base-layer packets explicitly because the decoder does not know a priori on which network layer they appear (i.e., the temporal layering algorithm moves the spatial base-layer packet up and down in the hierarchy). Accordingly, the PVH header contains a designated bit that is 1 for base-layer packets and is otherwise 0. Second, we must indicate how spatial layers are mapped onto network channels. For a given slice, we need to know which network layers contain actual data and which do not. We therefore explicitly encode these dependencies as a set of "resumption levels" in the base-layer packet that defines the slice. Finally, the decoder must know the specific arrangement of bits across layers in order to decode the bit stream. That is, the decoder must be able to switch layers dynamically during the decoding process as it encounters different segments of the spatial hierarchy. To do so, we prefix each block in the base layer with a special codeword called a *bit-allocation descriptor* (BD).

A BD indicates where in the hierarchy we encode the base-layer information and where each refinement pass appears for each of the three types of spatial components: DCT luminance coefficients, subband coefficients, and DCT chrominance coefficients. In effect, the BD codes the quantization information given earlier in Table 6.2. Because each image block has its own BD, we can carry out spatially adaptive quantization where some regions of the image have higher fidelity than others. To reduce the overhead of coding the BD's, the descriptor is spatially predicted. For example, we represent the BD with a single bit in the common case that it does not change from the previous image block.

Figure 6.11 illustrates the layout of the RTP/PVH packet header. In addition to the standard RTP header fields, the block ranges, and the base-layer bit mentioned above, the PVH header includes a version number and an EBIT field. Because packets are an integral number of bytes, some number of bits from the last octet should be discarded. The EBIT fields explicitly indicates this count. A PVH version number is included to incrementally deploy new versions of the codec. Also, if the packet is a base-layer packet (i.e., B is set), then an auxiliary header immediately follows the PVH header. This header includes the width and height (in blocks) of the video image as well as a count and list of the resumption levels and offsets described above. The appendix describes the exact syntax of these fields.

Figure 6.12 illustrates an example arrangement of packets in the slice-oriented hierarchy. Although coding layers are spread across network channels according to the temporal hierarchy, we simplify the diagram by indicating only the relationship among packets within the spatial hierarchy. Each labeled box corresponds to a packet header and the pair of numbers represents the range of macroblocks that are contained within the packet.

Each slice is identified by exactly one base-layer packet and the diagram contains two such slices, encircled by the dashed lines. Each base-layer packet additionally contains explicit pointers to all of the network channels that comprise the slice as indicated by the solid arrows. Moreover, each packet's resumption

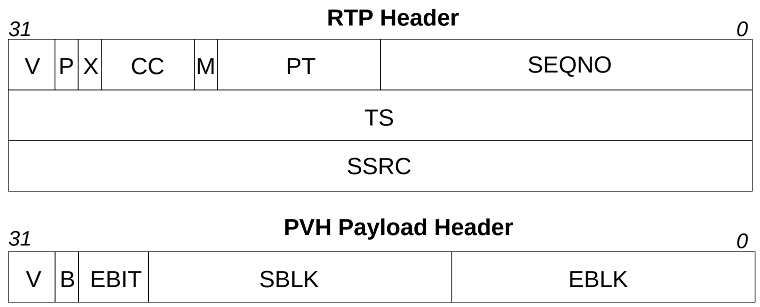


Figure 6.11: **RTP/PVH Packet Headers.** The RTP header contains a version number (V), a padding bit (P), an extension bit (X), a count of “contributing sources” (CC), i.e., for audio mixing or video compositing, a marker bit (M), a payload type (PT), a 16-bit sequence number (SEQNO), a media-specific timestamp (TS), and a “Source-ID” (SSRC). If the payload type indicates PVH, then a PVH header immediately follows the RTP header and consists of a PVH version number (V), a base-layer indicator (B), a count of padding bits (EBIT), a start block (SBLK), and an end block (EBLK).

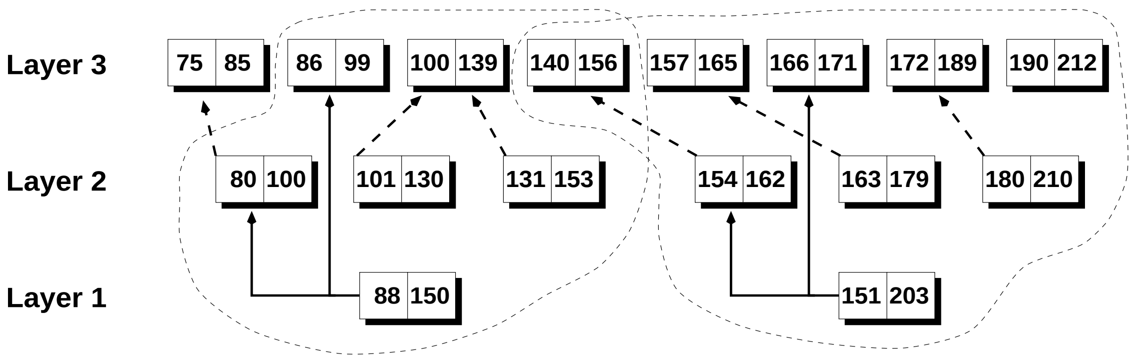


Figure 6.12: **Sample PVH Packet Stream.** Each base-layer packet defines a range of macroblocks that comprise a slice. Here, we show two slices, each enclosed by a dashed line, that are spread across the layer hierarchy.

pointer and offset is indicated by the dashed arrows. The packet that defines the (88,150) block range appears on layer 1 and naturally has its base-layer bit set (B=1). Each packet that is a member of the (88,150) slice is either wholly contained in or partially covers those blocks and is encircled by the lefthand dashed line. The base-layer packet additionally contains a count of resumption pointers and their values. For example, the base-layer packet points to successor packets in both layers 2 and 3, while the (101,130) layer 2 packet points to only the (100,130) layer 3 packet. If there were more layers, then the layer 2 packet would contain additional resumption pointers.

Given a base-layer packet, the decoder can extract the layer hierarchy and resumption pointers and offsets to definitively locate all the packets and layer offsets in a slice. A naive algorithm might perform this relatively complex task by buffering all received packets and scanning the buffer pool on each packet arrival to determine when slices become complete. Under this scheme, however, the decoder cannot easily differentiate between a packet that has not yet arrived and one that has been lost or reordered and hence cannot easily decide when to decode a partially received slice.

Instead of this data-driven approach to receiver buffering, we combine the timing recovery algorithm used by RTP-based applications with the slice reassembly algorithm. In this model, packets are synchronized across layers using the "playback point algorithm" modified to function across packet slices. That is, we schedule the packets from a given slice to be decoded together and discard the rare packet that arrives too late. When a slice's playback point arrives, we determine whether it is entirely intact and, if so, simply decode it. Otherwise, we invoke a loss recovery strategy to patch the missing data, possibly discarding unusable packets. (In practice, the loss recovery mechanism is folded into the decoding process.)

In our current implementation, we use the following hybrid of the data- and timer-driven approaches. We maintain two "containers" keyed by the RTP timestamp. Within each container, we maintain a circular buffer of packets for each layer and within each layer, we map packets directly into slots in a circular buffer using the low bits of the packet's RTP sequence number (so lookup and insertion are cheap). We also track the boundaries (i.e., the head and tail) of the current "window" of packets stored in a given layer. This allows us to quickly traverse over all the packets in a layer to check for gaps in the sequence space. Finally, we store all of the pending base-layer packets in a hash table for the current frame container (which typically is very sparse).

Whenever a base-layer packet arrives, we check whether its constituent slice is ready to decode by scanning each layer indicated in the resumption pointer list and checking if a contiguous block of packets at each layer "covers" the range of blocks in the base layer. If so, we decode the slice immediately and all packets wholly contained in the decoded slice are freed. Otherwise, the base layer packet is buffered and a timer is scheduled whose timeout is proportional to the packet interarrival time variance. If an enhancement layer packet arrives and completes the slice, then the slice is decoded and the timer is canceled. Otherwise if the timer expires, we assume packet loss occurred, invoke a loss recovery strategy, and decode the partial slice. When we are completely done with a frame, we free all the packets stored in the frame container data structure.

6.7 Summary

In this chapter, we presented our low-complexity layered codec targeted specifically for RLM and the environment in which RLM operates. Using the ALF/JSCC framework proposed in Chapter 4, we developed a codec that achieves temporal compression through multi-rate conditional replenishment and achieves spatial compression through a hybrid transform coding algorithm based on a split wavelet/DCT decomposition. In turn we developed a slice-based framing protocol that formats the output codewords directly into flexibly sized units for transmission on a best-effort packet network. Finally, we described our implementation that achieves run-time and compression performance on par with state of the art Internet codecs even while generating a layered representation.

Chapter 7

System Architecture

Although RLM and PVH fulfill a critical function for communication over heterogeneous networks, these two pieces provide just one component of an overall multimedia communication system — the video delivery component. We must therefore develop an overall system architecture that not only integrates RLM and PVH into an autonomous video application but also provides the functionality requisite to a complete multimedia communication system, including user-interface elements and companion applications like audio and shared whiteboard. Our "Composable Tools" framework is an agent-based approach for just such a system architecture. In this framework, we craft "media agents" from a common multimedia toolkit and control and configure them over a software interprocess communication bus that we call the *Coordination Bus*. By composing an arbitrary arrangement of media agents over the Coordination Bus and complementing the arrangement with an appropriate user-interface, we can induce an arbitrary multimedia collaboration style, e.g., an open meeting, moderated meeting, class, or seminar.

In this chapter, we describe the overall system architecture that underlies and integrates the MBone tools, focusing principally on the piece most relevant to PVH and RLM — the video conferencing application *vic*. We have developed this prototype application not only as an experimental platform for our codec and transmission scheme, but also as a useful communication tool in its own right. In designing *vic*, we had the benefit of hindsight gained through the MBone community's earlier experiences with the Xerox Network Video tool *nv* and the INRIA Videoconferencing System *ivs*. To encourage widespread acceptance of *vic*, we resolved to support backward compatibility with both *nv* and *ivs*, each of which used a different variant of an early version of RTP as well as a different compression format. This goal imposed design constraints that enhanced the flexibility of our resulting system.

Inevitably, in pioneering work such as *nv* and *ivs*, restrictions are imposed on the design process to facilitate experimentation. For example, the *ivs* design assumes video is represented as 4:1:1-decimated CCIR-601 YUV, while *nv* assumes 4:2:2 decimation. Extending *nv* to support H.261 or *ivs* to support the *nv* compression format would require non-trivial design changes. Also, since both systems are based on software compression, their video capture models were designed around uncompressed video. Extending either to support hardware-based compression engines would be relatively difficult.

Our tool *vic* builds upon the lessons learned from *ivs* and *nv* by focusing on flexibility. Not only did we develop a specific video conferencing application for our research vehicle but we developed a more general, re-usable multimedia programming environment. Underneath *vic*, we forged an extensible, object-oriented, application framework that supports:

- multiple network abstractions,

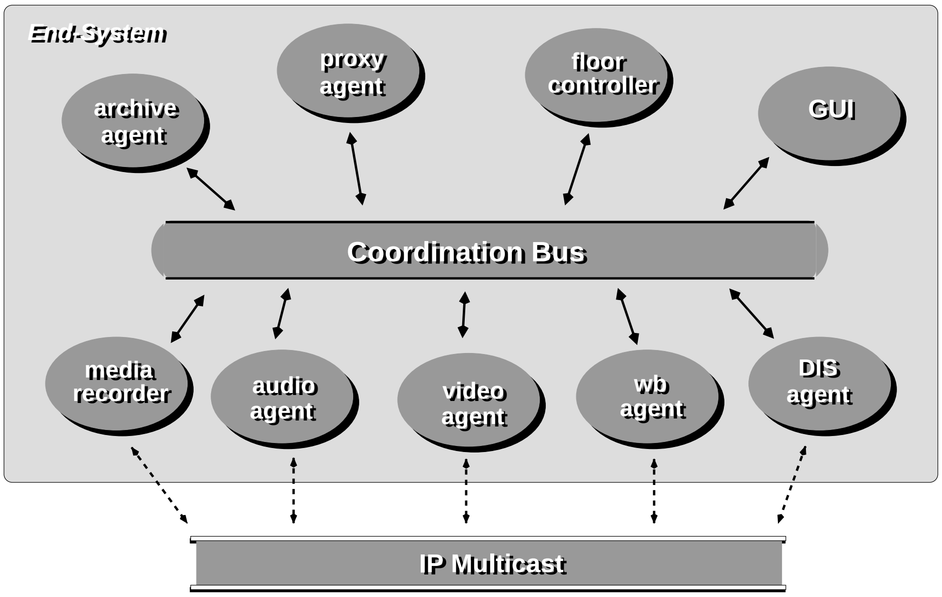


Figure 7.1: **The Coordination Bus.** The Coordination Bus is the backbone of an agent architecture where collaborative applications are built by composing specialized media agents.

- hardware-based codecs,
- a conference coordination model,
- an extensible user interface,
- diverse video compression algorithms, and
- a layered delivery model.

In the rest of this chapter, we describe both the architecture underlying *vic* as well as the overall system architecture that integrates the MBone tools. In the next section, we discuss our Composable Tools architecture and its core abstraction, the Coordination Bus. We then detail the multimedia toolkit design that we use to implement each composable element. Subsequently, we describe the software architecture of our layered transmission system, how layers are manipulated, and how the RLM/PVH modules interact. Next, we address one of the most performance-critical components of software-based video decoding: rendering and dithering of the video data from the YUV color space for display on an output device. Finally, we discuss the mechanisms for secure multimedia communication, user interface, and conclude.

7.1 The Composable Tools Framework

A cornerstone of the Unix design philosophy was to avoid supplying a separate application for every possible user task. Instead, simple, one-function "filters" like *grep* and *sort* can be easily and dynamically combined via a "pipe" operator to perform arbitrarily complex tasks. Similarly, we use modular, configurable applications, each specialized to implement a particular media, which can be easily composed via a *Coordination Bus* to support the variety of conferencing styles necessary for effective human communication. This approach derives from the framework proposed by Ousterhout in [130], where he claims that large systems are easily composed from small tools that are glued together with a simple communication primitive (e.g., the Tk *send* command). We have simply replaced his *send* primitive with a well-defined (and more restrictive) Coordination Bus protocol. By constraining the Coordination Bus protocol, we prevent the evolution of sets of tools that rely on the specifics of each other's internal implementations.

Figure 7.1 illustrates the high-level agent-based decomposition of our architecture. The concept is simple. Each application can broadcast a typed message on the bus and all applications that are registered to receive that message get a copy. Along the bottom of the diagram are a number of media agents that exchange real-time data flows over the network using IP Multicast. For example, the audio, video, and whiteboard agents might be implemented by our MBone tools: *vat* for audio, *vic* for video, and *wb* for shared whiteboard. Additionally, we might run a media recorder to archive the data and control streams within the session and the whole configuration might be used to implement a distributed simulation (DIS) and therefore involve a "DIS agent". Above the Coordination Bus we show a number of control agents that do not send data directly over the network but instead just manipulate other agents over the Bus. Our examples include a "floor controller" that might implement a floor control policy, an archive agent that might configure the media recorder and other agents over the Bus, and a proxy agent that might monitor the Bus' control activity in order to intelligently configure a remote media transcoder.

The notion of an agent-based architecture is not a new idea. At least one approach to collaborative design from the Artificial Intelligence community resembles our Composable Tools framework [48]. Edmonds et al. decompose their application functionality into a number of agents — user agents, groups floor agents, conference agents, presentation agents, and so forth — that interact over a software communication bus. In the context of the MBone tools, Handley and Wakeman independently developed the "Conference Control Channel Protocol" [76], which shares design goals and has architectural similarities with our approach. Likewise, Schulzrinne developed a coordination scheme that ties together the MBone tools using "Pattern Matching Multicast", a framework like our Coordination Bus with pattern-matching filters [151].

Our "Composable Tools" approach to networked multimedia contrasts with the more common "toolkit framework" adopted by other multimedia systems [36, 125, 143, 145]. Toolkits provide basic building blocks in the form of a code library with an application programming interface (API) to that library providing high-level abstractions for manipulating multimedia data flows. Each distinct conferencing style requires a different application but the applications are typically simple to write, consisting mostly of API calls with style-dependent glue and control logic.

The toolkit approach emphasizes the programming model and many elegant programming mechanisms have resulted from toolkit-related research. To simplify the programming model, toolkits usually assume that communication is application-independent and offer a generic, least-common-denominator network interface built using traditional transport protocols. In contrast, we emphasize the communication model by incorporating application level framing (ALF) not only at the network/application interface but also deep within the application. Under ALF, multimedia applications are simplified and both application and network performance enhanced by reflecting applications semantics in the network protocol. And ALF-based, media specific tools are a natural way to implement a simple solution to multimedia's biggest problem — high rate, high volume, continuous media data streams. Since the tools are directly involved in processing the multimedia data flows, we can use ALF to tune all the performance-critical multimedia data paths within the application and across the network.

In addition to performance, flexibility is gained by composing simple tools rather than using a monolithic application built on top of some API. Since each tool deals directly with its media stream and sends only low-rate reports like "user X is actively transmitting" on the Coordination Bus, the agent necessary to implement a particular conferencing scenario can be written in a simple interpreted language like *Tcl* [131]. This allows the most volatile part of the conferencing problem, the piece that melds audio, video, and other media into a coordinated unit that meets particular human needs and expectations, to be simple and easy to evolve. It also ensures that the coordination agents are designed orthogonal to the media agents, enforcing a mechanism/policy separation: media tools implement the mechanism by which coordination tools impose the policy structure appropriate for some particular conferencing scenario.

7.1.1 Light-weight Sessions, ALF, and RTP

Each of our composable tools is based not only on ALF but also on the Light-weight Sessions (LWS) model. In this framework, each media is implemented as a separate agent and the media are coordinated using side information disseminated on a control channel. LWS is simply a thin veneer over IP Multicast — because IP Multicast provides anonymity through group addresses, senders and receivers need not explicitly synchronize. A media source simply transmits data packets to the multicast group address. Audio and video data is transported via RTP while *wb* data is disseminated using a preliminary prototype of the Scalable Reliable Multicast (SRM) framework [59]. In [74], Handley and Crowcroft outline this overall architecture and relate the components to present and planned Internet standards.

Because of its ALF-like model, RTP is a natural match to our Composable Tools framework and serves as the foundation for the tools' network architecture. Complete details of the RTP specification are provided in [153] but we briefly review one feature of the protocol relevant to the Coordination Bus. Because media are distributed on independent RTP sessions (and because *vic* is implemented independently of other multimedia applications), the protocol must provide a mechanism for identifying relationships among media streams (e.g., for audio/video synchronization). Media sources are identified by a 32-bit RTP "source identifier" or Source-ID, which is guaranteed to be unique only within a single session. Thus, RTP defines a canonical-name identifier or CNAME that is globally unique across all sessions. The CNAME is a variable-length, ASCII string that can be algorithmically derived, e.g., from user and host names. RTCP control packets advertise the mapping between a given source's Source-ID and its variable-length CNAME. Using this relationship, a receiver can group distinct RTP sources via their CNAME into a single, logical entity that represents a given session participant.

This model — where each media is implemented by a separate agent running on a distinct RTP session (or set of sessions) all orchestrated across the Coordination Bus — requires a mechanism for determining the session address mappings, launching the tools, and connecting the agents over the Coordination Bus. In the current design, all of this functionality resides in a session advertisement tool like *sdr* [73]. As described in Chapter 3, *sdr* implements the Session Description Protocol to automatically advertise sessions and related mappings for individual media to multicast addresses. In addition, *sdr* consolidates these mappings, presents their corresponding high-level session name to the user, and allows the user to invoke the session. When the user starts the session, *sdr* launches and configures each separate agent according to the original advertisement and additionally attaches each agent to a shared Coordination Bus using a common bus address.

7.1.2 The Coordination Bus

Once *sdr* instantiates all the media agents for a given collaboration and ties them together with the Coordination Bus address, the agents can then interact using the Coordination Bus protocol to generate high-level functionality realizable only through their concerted organization. A detailed description of the Coordination Bus architecture is outside the scope of this thesis; rather, we provide a brief overview of the mechanisms in *vic* that support this model.

Voice-switched Windows

A feature not present in the other MBone video tools is *vic's* voice-switched windows. A window in voice-switched mode uses cues from *vat* to focus on the current speaker. "Focus" messages are broadcast by *vat* over the Coordination Bus, indicating the RTP CNAME of the current speaker. *Vic* monitors these messages and switches the viewing window to that person. If there are multiple voice-switched windows, the most recent speakers' video streams are shown. Because the focus messages are broadcast to any interested agent over the Coordination Bus, other applications can use them for other purposes. For example, on a network that supports different qualities of service, a QoS tool might use the focus message to give more video bandwidth to the current speaker, e.g., using dynamic RSVP filters [17]. Alternatively, a video gateway might monitor the audio channel to dynamically adapt the allocation of video bandwidth to each source in the session. For example, the gateway could allocate the largest share of bandwidth to the most recently active user and small amounts of bandwidth to less recently active users [4].

Floor Control

All of our MBone tools have the ability to "mute" or ignore a network media source, and the disposition of this mute control can be controlled by external agents via the Coordination Bus. This very simple mechanism provides a means to implement floor control in an auxiliary tool by composing functionality across multiple tools rather than adding new functionality to existing tools. One possible model is that each participant in the session follows the direction of a well-known (session-defined) moderator. The moderator grants the floor to some participant by multicasting a *takes-floor* directive over a multicast control channel indicating that participant's RTP CNAME. In response, each receiver locally mutes all participants except the one that holds the floor. Note that this model does not rely on cooperation among all the remote participants in a session — a misbehaving participant is prevented from disrupting the session because it is otherwise muted by all conforming participants.

Synchronization

Cross-media synchronization can also be carried out over the Coordination Bus. Each real-time application induces a buffering delay, called the *playback point*, to adapt to packet delay variations [88]. We can broaden the semantics of the playback point to effect synchronization across media. By broadcasting "synchronize" messages across the Coordination Bus, the different media can compute the maximum of all advertised playout delays. This maximum is then used in the delay-adaptation algorithm. In order to assure accurate synchronization, the semantics of the advertised playback points must be the delay offset between the source timestamp and the time the media is actually transduced to the analog domain (i.e., receiver buffering delay alone does not capture the local delay variability among codecs). Kouvelas et al. implemented a lip synchronization technique based on this architecture using a modified version of *vic* and their audio tool *rat* [105].

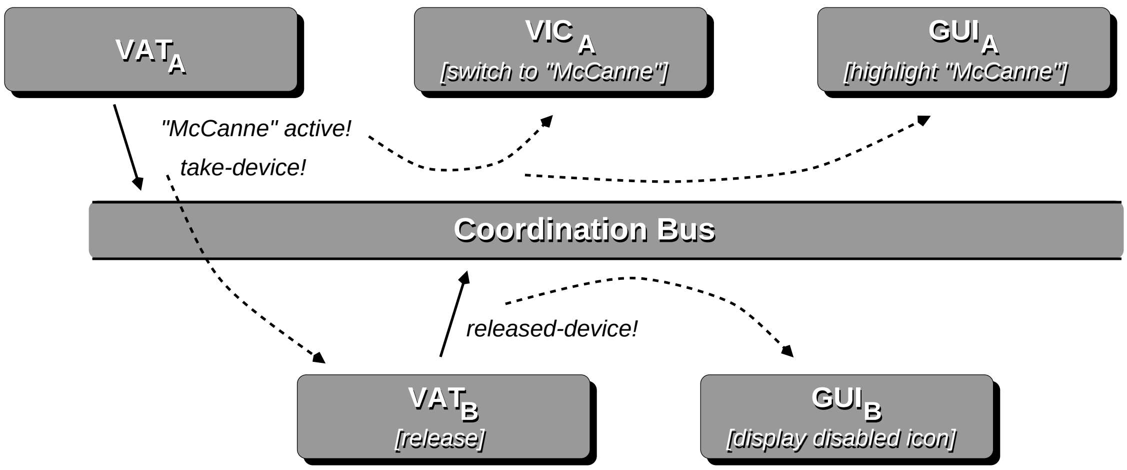


Figure 7.2: **Coordination Bus Event Flow.** When a new participant takes the floor in an idle session (A), a sequence of events signaled over the Coordination Bus causes (1) the audio device to be transferred to the active session, (2) the video to switch to the active speaker, and (3) the user-interface to highlight the corresponding participant.

Device Access

Each active session has a separate Coordination Bus to coordinate the media within that session. But some coordination operations like device access require interaction among different sessions. To this end, we define a global Coordination Bus, in addition to the session buses, that is shared among all media. For example, applications coordinate ownership of exclusive-access devices by exchanging *claim-device* and *release-device* messages over the global bus.

Implementation

We implement Coordination Buses as multicast datagram sockets bound to the loopback interface. Local-machine IP multicast provides a simple, efficient way for one process to send information to an arbitrary set of processes without needing to have the destinations "wired in". Since one user may be participating in several sessions simultaneously, the transport address (UDP destination port) is used to create a separate bus for each active session. This simplifies the communication model since a tool knows that everything it sends and receives over the bus refers to the session it is participating in and also improves performance since tools are awakened only when there is activity in their session. Each application in the session is configured with the address (port) of its bus via a start-up command line argument. The global device access bus uses a reserved port known to all applications.

Example

Figure 7.2 illustrates a distributed activity effected among a collection of agents communicating over the Coordination Bus. In this example, each box represents an agent, which might be implemented as an independent process or thread, and all of the agents run on the same end-system. The agents at the top of the diagram belong to one session (A), while the agents at the bottom belong to another (B). Session-A is

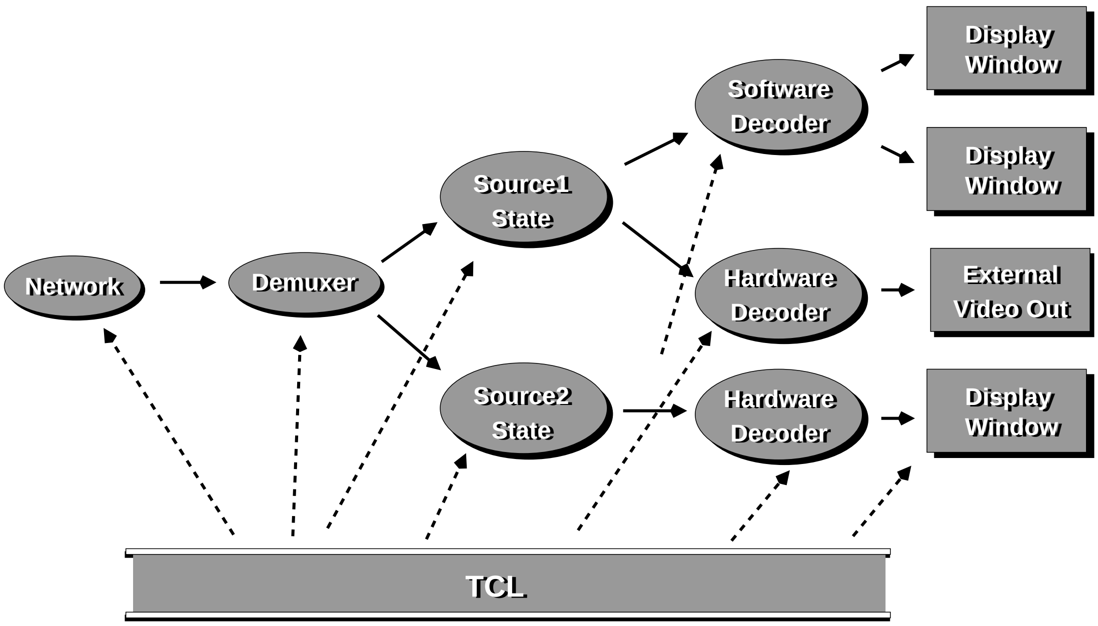


Figure 7.3: **The Receive/Decode Data Path.** Here we illustrate the interaction between the Tcl control path and the highly-optimized multimedia data paths. Dashed lines represent low-rate control flow while solid lines represent high-rate, performance-critical data flow. Tcl creates, configures, and composes objects that, once in place, efficiently process multimedia data all the way from the network through software based codecs to the user display.

composed of audio and video, while Session-B has only an audio channel. Assume initially that Session-B is active and therefore that the lower *vat* holds the exclusive-access audio device. At some point, user "McCanne" begins speaking on the A audio channel. Because the upper *vat* does not have the audio channel, it broadcasts a *take-device!* directive over the Coordination Bus, requesting the owner of the audio device to relinquish it. The lower *vat* sees the request, releases the device (as indicated in the diagram by the bracketed action), and broadcasts its *released-device!* action on the Bus. (Note that if Session-B were active, it might have decided against relinquishing the device.) Upon learning that the device is released, the Session-A *vat* opens the device and begins playing out the audio. Simultaneously, it broadcasts a message over the Bus indicating that user "McCanne" is now active. In turn, this message is picked up both by the video agent *vic* and by the Session-A user-interface agent. The user-interface responds by highlighting the active participant's name in the display, while *vic* responds by switching a video window that is configured in "voice-switched mode" to the new speaker. Similarly, upon noting that the Session-B *vat* released the audio device, the Session-B user-interface displays its audio device icon in an inactive format.

Although we have developed a preliminary version of our Coordination Bus protocol and successfully used it for a number of orchestration mechanisms, we have not refined it and documented it to the point that independent developers can easily build on top of it. Also, the user-interface/agent split implied by the architecture has not yet been fully implemented. These areas are topics of future work.

7.2 Software Architecture

Each media agent in our Composable Tools framework is itself built using an object-oriented software architecture. Our central goal was to create a flexible and extensible software framework to explore new coding schemes, network models, compression hardware, and conference control abstractions. By building on an object-oriented ALF framework, we achieved this flexibility without compromising the implementation's efficiency.

ALF leads to a design where data sources and sinks within the application are highly aware of how data must be represented for network transmission. For example, the software H.261 encoder does not produce a bit stream that is in turn packetized by an RTP agent. Instead, the encoder builds the packet stream fragmented at boundaries that are optimized for the semantics of H.261. Likewise, the PVH coder is designed specifically for a RTP-based packet transmission and the encoder explicitly frames coding-units into packets. In this way, the compressed bit stream can be made more robust to packet loss.

At the macroscopic level, the software architecture is built upon an event-driven model with highly optimized data paths glued together and controlled by a flexible Tcl/Tk [131] framework. A number of basic objects are implemented in C++ and are coordinated via Tcl/Tk. Portions of the C++ object hierarchy mirror a set of object-oriented Tcl commands. C++ base classes permit Tcl to manipulate objects and orchestrate data paths using a uniform interface, while derived subclasses support specific instances of capture devices, display types, decoder modules, and so forth. This division of low-overhead control functionality implemented in Tcl and performance-critical data handling implemented in C++ allows for rapid prototyping without sacrifice of performance.

7.2.1 Decode Path

Figure 7.3 roughly illustrates the receive/decode path. The elliptical nodes correspond to C++ base classes in the implementation, while the rectangular nodes represent output devices. A Tcl script constructs the data paths and performs out-of-band control that might result from network events or local user interaction. Since Tcl/Tk also contains the user interface, it is easy to present control functionality to the user as a single interface element that might invoke several primitive control functions to implement its functionality.

The data flow through the receive path is indicated by the solid arrows. When a packet arrives from the network, the Network object dispatches it to the Demuxer, which implements the bulk of the RTP processing. From there, the packet is demultiplexed to the appropriate Source object, which represents a specific, active transmitter in the multicast session. If no Source object exists for the incoming packet, an up-call into Tcl is made to instantiate a new data path for that source. Once the data path is established, packets flow from the source object to a decoder object. Hardware and software decoding, as well as multiple compression formats, are simultaneously supported via an object class hierarchy. When a decoder object decodes a complete frame, it invokes a rendering object to display the frame on the output device, either an X Window or an external video output port.

Note that, in line with ALF, packets flow all the way to the decoder object more or less intact. The decoder modules are not isolated from the network issues. In fact, it is exactly these modules that know best what to do when faced with packet loss or reordering. Object inheritance provides a convenient mechanism for implementing an ALF model without sacrificing software modularity.

While this architecture appears straightforward to implement, in practice the decode path has been one of the most challenging (and most revised) aspects of the design. The core difficulty is managing the combinatorics of all possible configurations. Many input compression formats are supported, and deciding the best way to decode any given stream depends on user input, the capabilities of the available hardware, and the parameters of the video stream. For example, Digital's Sound and Motion J300 adaptor supports hardware decompression of 4:2:2-decimated JPEG, either to an X Window or an external output port. The board can be multiplexed between capture and decoding to a window but not between capture and decoding to the external port. Also, if the incoming JPEG stream is 4:1:1 rather than 4:2:2-decimated, the hardware cannot be used at all. Finally, only JPEG-compressed streams can be displayed on the video output port since the system software does not support a direct path for uncompressed video. Many other devices exhibit similar peculiarities.

Coping with all hardware peculiarities requires building a rule set describing legal data paths. These rules depend intimately on how the application is being used, and therefore are complicated by user configuration. We have found that the Tcl/C++ combination provides a flexible solution for this problem. By implementing only the bare essentials in C++ and exporting a Tcl interface that allows easy creation, deletion, and configuration of C++ objects, the difficulty in managing the complexity of the data paths is greatly reduced.

7.2.2 Capture Path

We applied a similar architectural decomposition to the video capture/compression path. As with the decoder objects, encoder objects perform both compression and RTP packetization. For example, the H.261 encoder fragments its bit stream into packets on "macroblock" boundaries to minimize the impact of packet loss.

Different compression schemes require different video input formats. For instance, H.261 requires 4:1:1-decimated CIF format video while the *nv* encoder requires 4:2:2-decimated video of arbitrary geometry. One implementation approach would be for each capture device to export a common format that is subsequently converted to the format desired by the encoder. Unfortunately, manipulating video data in the uncompressed domain is a substantial performance bottleneck, so we have optimized the capture path by supporting each format.

A further performance gain was realized by carrying out the "conditional replenishment" step as early as possible (see § 6.2). Most of the compression schemes utilize block-based conditional replenishment, where the input image is divided up into small (e.g., 8x8) blocks and only blocks that change are coded and sent. The send decision for a block depends on only a small (dynamically varying) selection of pixels of that block. Hence, if the send decision is folded in with the capture I/O process, most of the read memory traffic and all of the write memory traffic is avoided when a block is not coded.

7.2.3 The Toolkit: MBTK

In this section, we provide a number of details on the Tcl/C++ object-oriented architecture that underlies *vic* and *vat*. We call this code library the "MBone Toolkit" or MBTK[1]. Over the past few years, this software architecture has evolved considerably and we plan to modify it in the near future. This discussion refers to the architecture that is in place in version 2.8 of *vic* (and version "4.0alpha" of *vat*). We assume a working knowledge of C++ and Tcl.

As in VuSystem and CMT, the core model of MBTK is object-oriented — objects produce, transform, or consume multimedia data. In all of these systems, objects implement highly-optimized multimedia

[1]MBTK is not currently a stand-alone toolkit but is instead embedded into the *vic/vat* implementation. Future work at U.C. Berkeley will factor out this toolkit into reusable library.

data paths in a compiled language (C++ for MBTK and VuSystem and C for CMT) while the objects themselves are composed, linked together, and configured via an interpreted language (Tcl in each case). The VuSystem authors refer to the data path segments as *in-band* code and the control segments as *out-of-band* code. Unlike CMT, MBTK has no generic buffering model and unlike VuSystem, it (deliberately) lacks a uniform API between objects. Instead objects are optimized to their local environment by tailoring the intermodular API for the specific object interaction. Each object has an implicit type and only certain types of objects can be connected to others. Because there is no explicit type information, we cannot perform run-time type-checking. Hence, misbehaving Tcl programs often cause hard-to-debug run-time faults and aborts rather than graceful, easy-to-debug error messages. (We plan to add object type-checking to a future version of MBTK.)

In the next two sections, we describe the Tcl/C++ object architecture. We first describe the Tcl API to C++, then describe how we implement the stated Tcl object semantics. Rather than present a detailed description of MBTK, we simply describe the core mechanisms and give some illustrative examples to provide a flavor for the low-level software architecture.

MBTK Tcl Object API

The MBTK object framework extends the Tcl interpreter with just two Tcl commands: *new* and *delete*. As in C++, we use *new* to create objects and *delete* to destroy them. Not surprisingly, each Tcl object is mirrored by an underlying C++ object. The syntax for *new* is:

```
new $type [ $params ]
```

This command returns a unique name for a new object of type $*type* and as a side effect of object creation, the object instance appears explicitly as a named procedure in the Tcl interpreter (the procedure name is chosen identical to the object name). Generally, the name is stored in a Tcl variable and is never explicitly referenced. The single string argument $*params* is passed to the underlying constructor and provides additional arguments that may be needed at runtime. The brackets indicate that $*params* is optional and its format depends on the specific object type. Because the object name returned by *new* is identically a command registered with the Tcl interpreter, we can invoke a method on an object by simply calling the object name as a procedure. More conveniently, we view method invocation as having the following simple syntax:

```
$obj $method [ $arg0 $arg1 ...  ]
```

where *obj* is a Tcl variable that holds the object name and $*method* is the name of the called method with optional arguments as indicated. For example, we might create a video capture object, then create a specific encoder that carries out PVH compression, and finally attach them as follows:

```
set captureObject [new grabber vl]
set encoderObject [new encoder pvh]
$captureObject encoder $encoderObject
$captureObject start
```

We first create a grabber of type *vl*, e.g., corresponding to an SGI Video Library (VL) capture object. Then, we create an encoder for our layered format PVH. We point the grabber at the encoder and finally "start" the grabber to initiate frame capture. (This sequence of events is merely illustrative and oversimplifies the initialization that actually occurs to set up a video capture path.) Note that if we inadvertently attached the capture device to some other type of object, e.g., a PVH decoder, then a run-time fault would result when the grabber coerces the decoder object to an encoder object and invokes the non-existent *encode* method.

To destroy an object and free up resources associated with that object, we simply delete it with the *delete* command:

```
delete $obj
```

where *$obj* is the object name originally returned by *new*. For example, when the user disables the video transmission initialized above, we might execute the following code:

```
$captureObject stop
$encoderObject flush
delete $captureObject
unset captureObject
delete $encoderObject
unset encoderObject
```

We first halt the capture process, then flush the encoder object to ensure that the current frame has been fully transmitted over the network. Finally, we delete the underlying objects and "unset" the corresponding Tcl variables to maintain the invariant that each object instance variable is defined only when the shadowed C++ object exists.

MBTK C++ Object API

The aforementioned Tcl object structures are implemented by mirroring each Tcl object with an underlying C++ object. We export each C++ multimedia processing object into Tcl using the new/delete API from above and a uniform method invocation interface. All objects that appear in Tcl are rooted in a C++ base class called *TclObject*. The TclObject class has two methods relevant to our discussion: *name* and *command*. The *name* method is used by a C++ module when it needs access to the object's Tcl name, enabling method invocation from C++ to Tcl. For example,

```
Tcl::instance().evalf("%s activate %d", p->name(), id);
```

calls the activate method on the TclObject pointed to by *p* with the integer argument from the C++ variable *id*. (*Tcl::evalf* provides a convenient hook for invoking the Tcl interpreter with a *printf*-like syntax.)

We structure our uniform method invocation interface around the *TclObject::command* virtual method. Like all Tcl callbacks, the *command* method takes as input a count of arguments and their string values. A derived subclass of TclObject implements its object method API by filling in the virtual command method. For example, we structure the command method for the VL grabber roughly as follows:

```
int VLGrabber::command(int argc, const char** argv)
{
        if (strcmp(argv[1], "start") == 0) {
                start();
                return (TCL_OK);
        } else if (strcmp(argv[1], "...

        ...

        return (Grabber::command(argc, argv));
}
```

If this command method does not recognize the Tcl method dispatched here, it calls the command method of its immediate parent in the C++ class hierarchy. By doing so, we manually implement *method combination* — i.e., the invocation of a given method whose functionality is split across different classes in an inheritance hierarchy. For example, when the *start* method is invoked on a "vl grabber" object, this code catches the command, performs the action, and returns a success code to the Tcl interpreter. If the method is not matched, the command is passed to the immediate parent in the C++ class hierarchy, in this case to *Grabber::command*. If no class handles the method, the base class method, *TclObject::command*, ultimately gains control and returns an error message to the Tcl interpreter (which can be caught with the normal Tcl exception handling mechanisms). This style of explicit method combination contrasts with other object-oriented methodologies — e.g., the Common Lisp Object System or the Object Tcl programming model [175] — that automatically combine methods that are split across classes.

To facilitate the introduction of new object types into MBTK, we created a special base class called *Matcher* that provides a template for object creation. A programmer adds support for a new Tcl object class by creating a sub-class of a Matcher and defining its *match* method. The *match* method takes as an argument the Tcl object class name and returns a new C++ instance of that object if it recognizes the name. Otherwise, it returns a null pointer. Using this simple API, the Tcl *new* command invokes the *match* method on each known Matcher class until a match is found. Once found, it returns the name of the new object (using the *TclObject::name* method) to the Tcl interpreter, which causes that name to be returned to the caller from Tcl. Because the Matcher constructor links each instance of a derived Matcher into a single linked-list, we can extend the object framework without modifying any of the existing source files by declaring a stub matcher instance staticly and linking in the new object module.

MBTK: Future Directions

Our original motivation for using the interpreted Tcl framework was to exploit the Tk user-interface toolkit and to leverage Tcl as a flexible glue for compositing and configuring our MBTK objects. Rather than develop a comprehensive object-oriented extension for Tcl, our focus instead was to simply export our C++ class hierarchy into Tcl. Because we did not intend to rely heavily on Tcl to structure the system, we did not foresee complexity arising in the Tcl implementation. Thus, our Tcl implementation was ad hoc, consisting of a small number of modules with little modularity or hierarchy. Over time, however, the convenience and ease of rapid-prototyping with an interepreted language encouraged us to migrate much of our tools' functionality into Tcl. This has led to substantial complexity in our Tcl scripts; we have thus concluded that we must revisit our Tcl-level design.

We plan to enhance MBTK with MIT's *Object Tcl* or OTcl [175]. On the one hand, our TclObject framework allows us to export C++ objects into a monolithic Tcl program, while on the other, OTcl provides an object-orienting programming methodology within Tcl itself. In future work, we intend to develop an object architecture across C++ and Tcl that provides a uniform Tcl API but allows methods to be implemented either in C++ or Tcl and further allows easy and transparent migration from Tcl to C++. For example, we might prototype a new method by implementing it in Tcl. If the object's run-time performance is ultimately inadequate, then we can migrate its method implementation into C++ without altering the API.

7.3 Application Support for Layered Streams

A detailed description of our entire software architecture is beyond the scope of this thesis. Instead, in this section, we describe one sophisticated use of the Tcl/C++ object interface in moderate detail to illustrate basic interaction among consituent objects. One rich piece of this architecture pariticularly relevant to this thesis is the interaction of the layered compression algorithm, the layered transmission system, and the underlying network system-call interface.

The key subsystems include:

- a *Network* object that implements the host's packet send/receive interface,
- an *RtpSession* object that implements RTP data demultiplexing and the RTCP control protocol,
- *Source* objects that represent each participant in an RtpSession,
- *SourceLayer* objects that each represent a single layer within a multi-layer RTP flow, and
- *PacketHandler* objects that decode packets from their compressed format for presentation to the user.

At start-up, the application creates an RTP session object and initializes the network by creating a Network object for each multicast data group and control group. If we assume that the Tcl array *address* contains the multicast addresses for each layer and the arrays *dataPort* and *ctrlPort* contain the data and control ports respectively, then the following code segment will build all the network objects and attach them to the session:

```
set session [new session rtp]
set i 0
while { $i <= $maxLevel } {
        set dataNet($i) [new network ip]
        $dataNet($i) open $address($i) $dataPort($i)
        $session data-net $i $dataNet($i)
        set ctrlNet($i) [new network ip]
        $ctrlNet($i) open $address($i) $ctrlPort($i)
        $session ctrl-net $i $ctrlNet($i)
        incr i
}
```

Note that we store the network objects in Tcl arrays so that we can access and manipulate them at a later time.

The interfaces among the network objects, the RTP session object, and RLM are all relatively simple. For example, the only action required between RLM and the network is for RLM to manipulate the number of layers received from each network source. In accordance with our Tcl/C++ model, RLM enables or disables a multicast group simply by invoking a method on the network object:

```
$net disable [$src]
$net enable [$src]
```

Because RLM is carried out on a per-source basis (i.e., congestion is a property of the path from a given source to the receiver), we explicitly indicate the source for which reception is enabled or disabled. If the *src* argument is omitted, then reception for the entire group is enabled or disabled. In the current implementation, we use this latter form because protocol machinery for source-based pruning has not yet been standardized nor has an extension to the application programming interface (e.g., the Berkeley socket API) been worked out.

For example, if we keep track of the current level of RLM subscription in the Tcl array *subscription* (indexed by the source object), then we might implement the procedure that adds a layer in an RLM join-experiment as follows:

```
proc add_layer src {
        global subscription maxLevel dataNet
        set currentLevel $subscription($src)
        incr currentLevel
        if { $currentLevel <= $maxLevel } {
                set subscription($src) $currentLevel
                $dataNet($currentLevel) enable $src
        }
}
```

Since the RLM protocol processing is not in the "fast path", run-time performance is not critical. Thus we implement RLM almost entirely in Tcl. The fast path packet processing code performs upcalls from C++ into Tcl when loss is detected to update the RLM loss estimators. In fact, we share the Tcl RLM implementation between *vic* and our simulator *ns*. By creating a well defined object API between RLM and the rest of the system, we can easily move the implementation between the simulation and real environments.

Packets are demultiplexed by the RtpSession object. When a packet arrives at the local host, the Network object gains control, reads the packet into a packet buffer, and passes the buffer to the RtpSession. The RtpSession notes which network object the packet arrived on and attaches the layer number to the packet buffer for later use by the decoder. It then consults a (hash) table of Source objects keyed on the RTP Source-ID. If no Source object exists, it performs an upcall into Tcl to allocate and install an object in the table[2]. After obtaining the Source object, the SourceLayer object is extracted from the Source using the network layer number. The SourceLayer carries out a number of RTP packet processing steps, and in particular when packets are lost, updates its loss rate estimator and calls Tcl to notify RLM of the loss. RLM then invokes its adaptation algorithm and possibly adjusts the application's level of subscription by performing downcalls back into C++ to enable or disable group membership.

After all generic RTP processing is completed, the packet is then passed to the corresponding Source object's PacketHandler. In the case of PVH, a PvhDecoder object — a derived subclass of PacketHandler — receives the packet and buffers it until an entire coding unit is present, as described in Chapter 6.

On the encoder side, each time a new frame becomes ready to code, the conditional replenishment module invokes the PVH encoder with the image data for the new frame, a vector representing the motion state of the block (see §6.2.2), and the temporal layer number. The encoder codes the blocks indicated by the motion information and formats the compressed output string into packets in different output layers according to the temporal layer number and spatial layer policies. Finally, the encoder delivers the packet buffer to the appropriate network object for physical transmission on the network.

[2] Actually, we do not activate a source until we are reasonably sure that the packet is from a valid RTP stream. We use a heuristic where we wait for two back-to-back packets and if the RTP sequence number changes by exactly one and all of the header fields are valid, then we accept the flow. Otherwise, we continue to ignore the stream. Without these checks, the application could become overwhelmed allocating an unending sequence of Source objects corresponding to a series of garbage packets, i.e., because the RTP Source-ID varies randomly. This might happen not only because an adversary attempts to disrupt the session but also when a benevolent user runs an encrypted session with an incorrect key.

7.4 Compression Formats

While *vic* supports the PVH scheme discussed above, it is also backward compatible with *nv* and supports several other formats. Our design philosophy for Internet video is that any bit stream produced by a sender must be decodable by any receiver. That is, even if a sender employs a hardware codec, all receivers must be able to decode the compressed stream in software. This means that we must implement a software decoder for each supported compression format. In addition to PVH, *vic* currently supports H.261, the *nv* format, Sun's CellB format, and Motion-JPEG.

The prevalence of Motion-JPEG hardware and widespread interest in using this medium over several high-speed testbeds motivated us to support hardware JPEG codecs in *vic*. Hence, *vic* must additionally be able to decode JPEG streams in software. However, the high data rates and temporal redundancy in Motion-JPEG lead to a computationally intensive decode process, which would perform poorly without tuning.

We applied several standard optimizations to our decoder (efficient DCT, inverse quantization folded with DCT, table driven Huffman decoding, minimization of memory traffic), but the most dramatic speedup was due to a novel computational pruning technique based on conditional replenishment. We maintain a reference cache of the six lowest frequency DCT coefficients of each block. As we decode a new block, we compare the reference coefficients against the newly decoded coefficients, and if the L_1 distance is below a configurable threshold, we skip the block entirely. Since JPEG does not carry out conditional replenishment in the compression algorithm itself, we apply conditional replenishment at the receiver to prune unnecessary computation. A similar thresholding algorithm is described in [68], though it is used for a different purpose (i.e., motion detection at the encoder).

7.5 Rendering

Another performance-critical operation is converting video from the YUV pixel representation used by most compression schemes to a format suitable for the output device. Since this rendering operation is performed after the decompression on uncompressed video, it can be a bottleneck and must be carefully implemented. Our profiles of *vic* match the experiences reported by Patel et al. [134], where image rendering sometimes accounts for 50% or more of the execution time.

Video output is rendered either through an output port on an external video device or to an X window. In the case of an X window, we might need to dither the output for a color-mapped display or simply convert YUV to RGB for a true-color display. Alternatively, HP's X server supports a "YUV visual" designed specifically for video and we can write YUV data directly to the X server. Again, we use a C++ class hierarchy to support all of these modes of operation and special-case the handling of 4:2:2 and 4:1:1-decimated video and scaling operations to maximize performance.

For color-mapped displays, *vic* supports several modes of dithering that trade off quality for computational efficiency. The default mode is a simple error-diffusion dither carried out in the YUV domain. Like the approach described in [134], we use table lookups for computing the error terms, but we use an improved algorithm for distributing color cells in the YUV color space. The color cells are chosen uniformly throughout the feasible set of colors in the YUV cube, rather than uniformly across the entire cube using saturation to find the closest feasible color. This approach effectively doubles the number of useful colors in the dither. Additionally, we add extra cells in the region of the color space that corresponds to flesh tones for better rendition of faces.

While the error-diffusion dither produces a relatively high quality image, it is computationally expensive. Hence, when performance is critical, a cheap, ordered dither is available. *Vic*'s ordered dither is an optimized version of the ordered dither from *nv*.

An even cheaper approach is to use direct color quantization. Here, a color gamut is optimized to the statistics of the displayed video and each pixel is quantized to the nearest color in the gamut. While this approach can produce banding artifacts from quantization noise, the quality is reasonable when the color map is chosen appropriately. *Vic* computes this color map using a static optimization explicitly invoked by the user. When the user clicks a button, a histogram of colors computed across all active display windows is fed into Heckbert's median cut algorithm [79]. The resulting color map is then downloaded into the rendering module. Since median cut is a compute-intensive operation that can take several seconds, it runs asynchronously in a separate process. We have found that this approach is qualitatively well matched to LCD color displays found on laptop PCs. The Heckbert color map optimization can also be used in tandem with the error diffusion algorithm. By concentrating color cells according to the input distribution, the dither color variance is reduced and quality increased.

Finally, we optimized the true-color rendering case. Here, the problem is simply to convert pixels from the YUV color space to RGB. Typically, this involves a linear transformation requiring four scalar multiplications and six conditionals. Inspired by the approach in [134], *vic* uses an algorithm that gives full 24-bit resolution using a single table lookup on each U-V chrominance pair and performs all the saturation checks in parallel. The trick is to leverage off the fact that the three coefficients of the Y term are all 1 in the linear transform. Thus we can precompute all conversions for the tuple $(0, U, V)$ using a 64KB lookup table, T. Then, by linearity, the conversion is simply $(R, G, B) = (Y, Y, Y) + T(U, V)$.

A final rendering optimization is to dither only the regions of the image that change. Each decoder keeps track of the blocks that are updated in each frame and renders only those blocks. Pixels are rendered into a buffer shared between the X server and the application so that only a single data copy is needed to update the display with a new video frame. Moreover, this copy is optimized by limiting it to a bounding box formed across all the updated blocks of the new frame.

7.6 Privacy

To provide confidentiality to a session, *vic* implements end-to-end encryption per the RTP specification. Rather than rely on access controls (e.g., scope control in IP Multicast), the end-to-end model assumes that the network can be easily tapped and thus enlists encryption to prevent unwanted receivers from interpreting the transmission. In a private session, *vic* encrypts all packets as the last step in the transmission path, and decrypts everything as the first step in the reception path. The encryption key is specified to the session participants via some external, secure distribution mechanism.

Vic supports multiple encryption schemes with a C++ class hierarchy. By default, we use the Data Encryption Standard (DES) in cipher block chaining mode [6]. While weaker forms of encryption could be used (e.g., those based on linear feedback shift registers), efficient implementations of the DES give good performance on current hardware (measurements are given in [91]). The computational requirements of compression/decompression far outweigh the cost of encryption/decryption.

One attractive feature of layered streams is that we can accommodate heterogeneity even with encryption because we can simply encrypt each layer of a hierarchical flow independently. Since our layered transmission system adapts only by adjusting the number of layers delivered, encryption imposes no difficulty. In contrast, systems that use transcoding or some other syntactic manipulation of the media flow to adjust the rate within the network cannot operate on encrypted streams without knowledge of the encryption

Figure 7.4: **Vic's User Interface.** The video-only user interface consists of three window types: a thumbnail listing in the upper left corner, a control panel in the lower right corner, and viewing windows.

key. Thus, complete end-to-end encryption, while trivial for RLM/PVH, is infeasible for transcoder-based bandwidth adaptation systems.

7.7 User Interface

A screen dump of *vic's* user interface, illustrating the display of several active video streams, is shown in Figure 7.4. The main conference window is in the upper left hand corner. It shows a thumbnail view of each active source next to various identification and reception information. The three viewing windows were opened by clicking on their respective thumbnails. The control menu is shown at the lower right. Buttons in this window turn transmission on and off, select the encoding format and image size, and give access to capture device-specific features like selection among several video input connectors. Bandwidth and frame rate sliders limit the rate of the transmission, while a generic quality slider trades off quality for bandwidth in a fashion dependent on the selected encoding format.

Because this user interface is implemented as a Tcl/Tk script embedded in *vic*, it is easy to prototype changes and evolve the interface. Moreover, the interface is extensible since at run-time a user can include additional code to modify the core interface via a home directory "dot file".

A serious deficiency in our current approach is that *vic*'s user interface is completely independent of the other media tools in the session. While this modularity is fundamental to our system architecture, it can be detrimental to the user interface and a more uniform presentation is needed. For example, *vat*, *vic*, and *wb* all employ their own user interface element to display the members of the session. But clearly a better model is to have a single instance of this list across all the tools. We intend to evolve our tools in this direction by merging the user interfaces of the different tools into an integrated interface that plugs into the Coordination Bus. Different application styles and arrangements could be easily implemented as separate programs, using the scripting language to realize the user interface and orchestration.

7.8 Summary

In this chapter, we described the network and software architectures of *vic*. By building the design around application-level framing, we achieved a highly flexible decomposition without sacrificing performance. A key benefit of this flexible framework is the ability to experiment with new video coding algorithms, which we exploited with the development of PVH and Intra-H.261. By applying elements of the *nv* codec design to the traditional H.261 compression algorithm and to our PVH design, we produced new coding schemes that balance the tradeoff between good compression gain and packet loss resilience. Moreover, by leveraging upon RTP, we benefit from a solid, well-defined protocol framework that promotes application interoperability, while maintaining the ALF philosophy that minimally restricts application design and, in particular, leads to an efficient implementation. Finally, we described our approach to building networked multimedia configurations out of composable tools, and the mechanisms we deployed in *vic* to support this composable architecture via the "Coordination Bus".

Chapter 8

Conclusions and Future Work

We close this thesis with an enumeration of several remaining challenges to our proposed approach. We then present a number of research problems that may be addressed in future work. Next we describe several cases where both RLM and PVH may potentially impact research areas outside of our layered video architecture. Finally, we outline the availability of our reference implementations, simulation scripts, and tools, and conclude.

8.1 Outstanding Problems

Although the RLM/PVH framework provides a promising foundation for scalable delivery of video in heterogeneous environments, a number of outstanding problems must be addressed to deploy the system over the current and future Internet MBone:

- **Prune-state Scaling.** RLM relies on routers to selectively prune flows using per-source group membership state. Under the current IP Multicast architecture, the amount of state that must be managed in the routers scales in proportion to the product of the number of groups and the number of sources. One way to improve this scaling behavior is to use a centralized rather than source-based approach to multicast routing as in CBT [7] and Sparse-mode PIM [40]. Although this approach improves the state requirement to scale linearly with the number of groups, it precludes the efficiency of shortest path routes in a source-based tree. Furthermore, the scale of future networks will likely require *sub-linear* growth and no current approach offers this degree of performance. Such a solution is likely to involve hierarchy and aggregation of prune state across network administrative boundaries, but this remains an active area of open research.
- **Source-based Prunes.** In RLM, when a receiver starts up and joins the base group of a session, it receives every base layer transmission of every active stream. In certain environments, this turn-on transient can be large enough to disrupt a large subset of the session. If the receiver could instead learn about the set of active sources before subscribing to their flows, it could gracefully join the session by picking and choosing among active flows. Additionally, the receiver might use side information from a conference control protocol to decide which sources are more interesting and subscribe to those sources first. To this end, we could distribute only control information (at a scalable low rate) on the first multicast group and distribute each video layer on the subsequent multicast groups. Receivers could then use explicit knowledge of existing sources to join multicast groups on a per-source basis.

Although this mechanism has not yet been worked out and deployed, a new version of the host group membership protocol that includes such capabilities (IGMP-3) is currently under review by the IETF.

- **Receiver Consensus.** An important requirement of RLM is that all users cooperate. The bandwidth utilization of any given link is determined by consensus among all of the participants downstream from that link. If just one user in a large group defects and joins all the layers, then nothing can be done to counteract the resulting congestion. Of course, if everyone is running RLM, this will not happen. On the other hand, given the way multicast membership is managed and how RLM might be implemented, more subtle failure modes are possible. For example, a user might manually "suspend" an RLM-based application, preventing the end-host from reacting to future congestion. One solution to this problem is to modify the operating system to disable group membership for suspended processes and (re-)enable membership when the process is resumed[1].

- **Protocol Interaction.** Although we examined several simulation configurations that explored the interaction of RLM with other protocols, we have not studied protocol interdependencies in great detail. Because of the non-linear, non-stationary, and often unpredictable nature of real protocol implementations, analysis of their interaction is difficult and poorly understood. We attempted an initial characterization of these interactions through some rudimentary simulation work, but it is not clear that RLM will interact gracefully with other congestion control schemes at very large scale or in arbitrary configuration. Modeling the interaction of protocol control algorithms and bounding their behavior under realistic conditions is a difficult and open problem.

8.2 Future Work

In the next two sections, we identify several new research areas that could potentially improve the performance of both RLM and PVH.

8.2.1 RLM

In this section, we outline a number of areas for future work related to RLM:

- **Load vs. Loss Adaptation.** One of the disadvantages of software-based compression schemes is their reliance on computational resources of the local host. If a hardware codec or a high-end workstation sources a high-rate video stream, a low-end host might not be able to decode and render every frame in the stream. In this case, packets are dropped by the operating system due to input buffer overflows and quality degrades dramatically. To address this problem, we can exploit RLM to gracefully adapt the application to the available CPU resources by shedding load [51, 34].

  On the one hand, RLM provides a convenient mechanism for adapting not only to network loss but also to local computational resources because its reaction of dropping layers reduces the local processing burden. Hence, we can carry out both load adaptation and network congestion adaptation exclusively with RLM. On the other hand, if we distinguish between network loss and local loss and separate the network and computational adaptation schemes, we can achieve improved performance because the two schemes should react over different time scales.

[1] On many Unix systems, the application can catch a user-initiated suspension event and drop all layers. Likewise, the application can catch the resumption event, restore the groups, and continue normal operation.

For example, two video applications might be competing for the CPU and when one of them exits, an excess of local processing resource will suddenly appear to the other. In response, this application should add layers because it now has the computational capacity to decode at a higher level of quality. We could either rely on RLM to eventually probe for new layers (which will now succeed since the processor can again keep pace with the incoming data), or we could explicitly notice that the local environment has changed and react by adding the previously dropped layers. The system could simply remember that layers had been previously shed due to a past computational deficit that is no longer relevant.

- **Generalized Layering.** While RLM/PVH operates principally under the assumption that additional layers provide improved video quality, layers can be formed across other dimensions. One alternative layering structure is to assign the different active sources in a given session dynamically to the different multicast groups using a loosely coupled, distributed consensus protocol. For example, a conference control protocol might determine which source holds the floor and in turn direct that source to send on layer 1. Other less recently active sources are in turn demoted to higher layers. In this fashion, bandwidth constrained receivers that run the normal RLM protocol receive video from only the participant who holds the floor while high-rate receivers receive all active flows. Moreover, layered encodings can still be exploited by assigning one source to one set of layers and another source to another. Since the mapping is arbitrary, we might put the base layer of source A on layer 1 the base layer of source B on layer 2, the enhancement layer of both source A and source B on layer 3, and so on. Finally, this scheme could be extended across arbitrary media, where we inter-mix audio, video, text, and graphic data (and layered representations of each) dynamically over a set of RLM-controlled multicast groups.

- **Tentative Joins.** Further work is needed to characterize and refine the interaction among RLM, RED, and the tentative-graft extension described in §5.2. We have not yet implemented nor simulated the performance of this proposed mechanism. An important component of a RED gateway is the algorithm that decides when a flow is non-responsive and therefore should be penalized by preferentially discarding its traffic [56]. This "penalty box" classification scheme could potentially subsume the RLM tentative-graft reaction. Alternatively, the two algorithms could share a common implementation but be individually tailored through configuration parameters.

8.2.2 PVH

In this section, we outline a number of areas for future work related to PVH:

- **Codec Performance.** Our PVH codec implementation still lacks refined maturity and its performance can be further improved in several ways:

  - First, several of the Huffman tables were designed either in an ad hoc fashion, by hand, or borrowed from existing codecs. Compression performance could be improved through a systematic analysis that determined better default tables. Moreover, because the statistics of each signal layer vary, the tables should vary according to the layer number to better reflect these statistics and thereby achieve higher compression. Currently, the tables are fixed across all layers.

    While the default tables can be optimized staticly, we can additionally improve performance by running a dynamic adaptation algorithm to tailor the Huffman table for the live signal's statistics. Under this scheme, we run an on-line adaptation algorithm that tracks signal statistics and disseminates a table update whenever the statistics are sufficiently different to warrant a new table.

A very simple approach is to simply run the optimization once at start-up (or manually triggered by the user) for a small sample. If the statistics do not vary extensively (e.g., as in a typical conference room or classroom setting), then this simple optimization would prove effective.

- Second, we can improve compression performance of several aspects of the algorithm with some minor modifications. For example, the refinement information for each of the four subband coefficients in a quad-tree leaf is coded independently. But there is correlation to exploit in the quad-tree leaf (e.g., many such quad-trees consist of multiple zeros) and jointly coding the four coefficients should give better performance. This vector quantization step can be carried out efficiently with 8-bit table lookup composed of the four refinement bits plus four context bits (that indicate whether each coefficient has previously been found to be significant).
- Third, we conjecture that table lookups can improve run-time performance in a number of other stages of the algorithm.

- **PVH Modeling.** Our simulations are currently limited by an inaccurate model of PVH; we need to develop better models to aid in performance evaluation. A model for layered signal sources that expresses the dependencies between packets in different layers will allow us to develop better loss metrics since losses in the core layers adversely impact higher layers. For instance, our simulations characterized performance based on aggregate loss behavior. By summarizing the performance in a single number, we implicitly assume that the layers are independent. On the other hand, a video model that accounted for the semantic dependence among layers would accurately reflect packet loss into the performance assessment.

- **Dynamic Rate-control.** Our PVH codec currently formats its output over a number of network channels according to a fixed set of progressive quantizers. But over time, the entropy of the underlying signal might vary dramatically (i.e., the signal becomes harder or easier to compress) causing the bit-rate of each output layer to change over time. Garrett and Willenger [65] have shown that typical video sources exhibit *long range dependence*, where rate behavior is bursty even on large time scales. However, our RLM simulations were carried out with video sources that assume a constant bit-rate output on each layer. Furthermore, RLM adaptation operates best under this model since dynamic variation in rates prevents the algorithm from converging to a stable operating point. If the video bandwidth varies faster than RLM can track it, then packet loss rates might never be reduced to an adequate level.

  One solution to this problem is to exploit *dynamic-rate control*. In this scheme, we adaptively adjust the quantization mix over the PVH codec layers to produce a constant bit-rate for each layer. To do so, we can exploit the fact that PVH is an instance of an *embedded code*, which has the property that any prefix of the compressed bitstream remains a valid representation at a lower quality. In other words, a given video frame can be successively refined at a fine-scale granularity. Using this property, we can partition the bit-rate arbitrarily among layers and vary this allocation dynamically, from frame to frame or slowly over time. Although PVH is amenable to this style of adaptation, our current implementation takes advantage of the fixed layout strategy to improve performance and consequently we have not yet developed such a rate allocation control strategy.

- **Feedback-driven Rate Partitioning.** We can similarly exploit the embedded property of PVH to adjust the bit-rate allocated to each layer based on feedback from the receivers. By monitoring scalable, low-rate feedback from RLM receivers (e.g., as provided by RTCP [153]), a source can tailor its overall rate allocation to the particular network environment. For example, if the entire session is connected at high-rate, but one user is connected at ISDN rate, the source could react by adjusting its layers to produce an exactly matched two-layer structure rather than a higher-complexity multi-layer stream.

- **Codec Synthesis.** A final area for future work related to PVH is a topic we call *automatic codec synthesis*. In this framework, a codec is specified as an interconnection of modules in a high level language and compiled into native code through translation and optimization. While earlier attempts to use high-level languages like C for signal processing algorithms have produced lukewarm results [161], we believe that better performance can be realized with a highly constrained specification syntax targeted specifically for video codecs and certain specialized optimizations[2]. Moreover, we can use this approach to effect dynamic code generation (DCG) [138] where the codec is synthesized on demand to incorporate run-time knowledge into the compile-time optimizations. For example, we can use DCG to optimize the fast path of a bottleneck operation like rendering. If we know the input and output image sizes at run time, we can compile code tailored specifically for that run-time configuration. Finally, this model replaces the accepted model of digital audio/video communication that is based on fixed, "standardized" compression formats with a dynamic framework, where coding algorithms can be specified in a high-level form and transmitted to the receiver. In this scheme, the codec need not be standardized since it is explicitly delivered to the receiver(s) for each conversation. A multi-profile system based on these concepts is currently under consideration by the MPEG-4 standards body[3].

8.3 RLM/PVH Beyond Layered Video

Concepts from both PVH and RLM can be applied to areas outside our layered video architecture. In an integrated services network, a receiver could explicitly negotiate with the network to determine the appropriate number of layers [83] with or without consideration of a pricing structure. Although RLM adaptation is not strictly necessary when the network provides resource guarantees, RLM might still be useful within this environment when resource management is coarse-grained. For example, Class Based Queuing (CBQ) [58] could be used to provide an "adaptive-rate video" traffic class with some specified bandwidth. Then within this CBQ class, independent video sessions would contend for the aggregate class bandwidth using RLM. The combination of CBQ and RLM can be used to isolate the interactions of end-to-end congestion control and thereby avoid unforeseen modes of controller instability.

The RLM framework could be combined with the Scalable Reliable Multicast (SRM) protocol framework [59] in the LBNL whiteboard *wb* to optimize the latency of rate-controlled transmissions. Because SRM uses a token-bucket rate-controller, it has the same limitations that single-layer video has in heterogeneous environments. On the other hand, several token-buckets with a range of rates could be used in tandem with multiple multicast groups and RLM. In this scheme, SRM would *simulcast* new data across all of the token-buckets to trade off bandwidth for latency. By spacing the rates exponentially, the overhead of the simulcast is minimized.

Like RLM, PVH has the potential to be incorporated into other environments outside our layered video architecture. One straightforward application is to use the PVH algorithm in a scalable image code by omitting the conditional replenishment syntax. Since the algorithm shares properties with Progressive-JPEG, its compression performance is comparable at medium and high rates, while at low rates it performs better

[2] This conjecture is based on our experience developing video codec optimizations under the constraints of the C++ programming language. Many times an interesting idea for an optimization was not pursued because of the difficulty in expressing it in C++.

[3] Unlike the MPEG-4 effort, we believe the requisite underlying technology — i.e., code generation and optimization techniques — should be developed before standardizing the virtual machine and run-time model.

because of its multiresolution subband decomposition. Hence, if PVH replaced Progressive-JPEG as the Web image compression scheme of choice, then performance of Web image transfers where its most noticeable — i.e., at low rate across dial-up lines — would be improved.

Likewise, both the PVH image and video formats have the potential to enhance the effectiveness and performance of Web proxies [60] and Web caches [16]. Rather than perform potentially heavy computation to transcode image formats on the fly, a bandwidth-adaptive proxy could selectively forward image or video layers to tailor the transfer to network path.

Finally, layered formats can be used in archive systems and digital libraries to facilitate manual browsing and searching of image and video data bases. A fully defined, implemented, and widely available coding scheme like PVH will facilitate the development of prototypes for applications like on-line digital libraries and archived university courses and seminars.

8.4 Status and Availability

Source code for *vic* has been publicly available since November 1994. Tens of thousands of retrievals of the software were made between the release date and the present. The system runs on all common workstation platforms as well as Windows 95/NT and the user community has continually ported *vic* to new environments.

Vic has been put to production use in several environments. An early version of *vic* was used by the Xunet research community to carry out distributed meetings in Fall 1993. Because bandwidth was plentiful, each site could (and did) transmit continuous video, placing the bottleneck in *vic*. This experience led to the voice-switched window feature and to the model by which streams are only fully decoded when being displayed.

Vic has been used in several class projects at U.C. Berkeley as well as in several external research projects. It was the test application in a study of the Tenet real-time protocols over the Sequoia 2000 network [9].

In November 1994, a live surgery performed at the U.C. San Francisco Medical School was transmitted to a medical conference in Europe using *vic's* Intra-H.261. During the surgical demonstration, the surgeon lectured to medical students at Middlesex and Whittington Hospitals in London and in Gothenburg, Sweden.

In Fall 1994, *vic* was used on the U.C. Berkeley campus to distribute course lectures over the campus network. The following spring, we equipped a seminar room for MBone transmission and have broadcast the Berkeley Multimedia and Graphics Seminar each semester since. In addition to our seminars, a number of other educational institutions have used *vic* for "distance learning" applications — the University of Colorado transmits their Computer Science colloquia, while the Berkeley Math Sciences Research Institute (MSRI) regularly broadcasts a seminar series. Several dissertation defenses and doctoral qualifying exams have been conducted over the network using *vic*. Some sites have begun experimental delivery of courses over the network and we believe that this type of course delivery will become commonplace in the next few years. For example, students might participate in and review lectures from their dorm rooms while outreach programs deliver continuing education to engineers in industry.

Finally, *vic* has been used to broadcast several standard MBone events, including NASA's live space shuttle coverage, the IETF meetings, and many technical conferences.

8.4.1 Status of Layered Architecture

The PVH codec, spatio-temporal layering, and RTP-based packetization scheme are all implemented in an experimental version of our video conferencing application *vic*. The PVH codec and framing protocol are implemented as a modular C++ object in the Tcl/Tk-based [131] multimedia toolkit described in Chapter 7. Likewise, a common RLM implementation is shared across our network simulator *ns* as well *vic*.

Even with RLM fully integrated into *vic*, the current framework is still experimental. We are just beginning to understand the interaction between RLM and other adaptive congestion control schemes, e.g., those in TCP/IP. Moreover, RLM requires the "fast leave" mechanism in IGMP to quickly react to network congestion, but this has not yet been widely deployed. Similarly, source-based pruning is not yet deployed or standardized.

While we continue to experiment with, refine, and deploy RLM, we can immediately leverage PVH by itself through the use of manually configured (hence nonscalable) distribution groups. Since IP multicast provides mechanisms to limit the "scope" of a group transmission, we can effect layered transmission though a hierarchical arrangement of scopes, where the layers in the distribution are allocated to a set of nested scopes each with a larger reach. That is, we can use distribution scope to topologically constrain the reach of each layer. For example, we might distribute the UCB MBone seminar by sending 32 kb/s to the "world" scope, 128 kb/s to the well-connected MBone, 256 kb/s across our campus network, and 1 Mb/s throughout the department network.

PVH can also be used in tandem with the Resource ReserVation Protocol (RSVP) [179], which supports the notion of layered reservations. In this approach, receivers negotiate explicitly with the network for bandwidth by adjusting their reservation to the maximum number of layers that the network can deliver [83].

Although transition from one technology to another is often a slow process — even in the MBone where new tools are deployed simply by distributing them over the network — the outlook for layered video is promising for several reasons:

- First, the extension of the RTP specification for layered streams will enable multiple, interoperable implementations.
- Second, the availability of a fast and efficient layered video codec (PVH) will bootstrap experimentation with layered media and demonstrate its ability to accommodate the Internet's heterogeneity.
- Finally, the large scale deployment of administrative scope mechanisms will enable the incremental deployment of layered transmission while we continue to refine the RLM framework.

We believe that these factors will combine to make layered video transmission commonplace in the Internet within the next few years.

8.4.2 Availability

All of the work contained in this thesis is open and our implementations are publicly available. A description of on-line resources, links to documentation and software, and pointers to related projects is available from:

`http://www.cs.berkeley.edu/~mccanne/phd/`

This URL includes links to our video application *vic*, the other MBone tools, our network simulator *ns*, simulation scripts that produced all of the data contained herein, the evolving documentation and standardization efforts for PVH, related work, and our follow-on research projects.

8.5 Summary

In this thesis, we proposed a framework for the transmission of layered signals over heterogeneous networks using receiver-driven adaptation. We evaluated the performance of our adaptation protocol RLM through simulation and showed that it exhibits reasonable loss and convergence rates under several scaling scenarios. We described the details of our low-complexity, loss-resilient layered source coder PVH and presented performance results to show that it performs as well as or better than current Internet video codecs. Moreover, the run-time performance of our software PVH codec is no worse than our highly tuned H.261 implementation (at equivalent signal quality) even though it produces a layered output format. Existing solutions to heterogeneous video transmission are either network-oriented or compression-oriented — in contrast, our focus is on complete system design and implementation. Together, RLM, PVH, and our system framework provide a comprehensive solution for scalable multicast video transmission in heterogeneous packet-switched networks like the Internet.

Appendix A

PVH Codec Version 1.0

(Draft 1.0α)

This appendix describes the bit-level syntax of version 1.0 of our "Progressive Video with Hybrid transform" (PVH) coder/decoder (codec). Although we attempt to provide a level of detail that is sufficient to implement a codec using only this document and without access to our reference implementation, this document has not yet been thoroughly reviewed nor are independent implementations yet realized. The current specification is in "draft status" and is likely to change in the near future as we improve the algorithm and clarify the text.

Version 1.0 of PVH is deliberately simple and omits a number of features in the interest of producing a low-complexity layered codec for flexible experimentation. Many parameters that are variable in other standards (e.g., DCT coefficient sample resolution and color downsampling ratio) are fixed in PVH. The selection of fixed parameters was generally chosen to favor good performance at low bit-rates and reduced complexity; thus, the highest attainable quality in the current codec is limited. Future versions may make these parameters configurable to increase the maximum attainable quality.

This document describes only a video coding algorithm and assumes that other components requisite to a complete multimedia communication system are defined elsewhere. The codec is based on an underlying packet-switched internetworks and utilizes the Real-time Transport Protocol (RTP) [153] and related standards from the Internet Engineering Task Force (IETF). In the RTP framework, a number of documents known as "payload format specifications" describe methods for adapting existing audio and video coding schemes for packet transmission using RTP. We diverge from this common practice of adapting existing codecs and instead propose a new video coding scheme targeted specifically for heterogeneous packet networks.

Unlike traditional video coding algorithms that assume a continuous bit stream coding model, the PVH codec assumes a packet-oriented coding model. Fundamental to the design of PVH is that the underlying network's packet structure is reflected into the coding structure for improved performance and better loss resilience. PVH does not utilize bit-level forward error correction coding (FEC) and instead leverages upon the fact that receipt of a packet implies that all bits within that packet are received without error (with very high probability). Although the design of PVH is deliberately loss-resilient, additional mechanisms for enhancing the reliability of a PVH packet stream (e.g., congestion control or packet-level FEC) are outside the scope of this document.

This document principally describes the decoding algorithm. Given a decoder specification, an encoding algorithm can be independently designed to generate output that conforms to the decoder syntax.

A.1 The PVH Model

PVH is a block-based video coder that produces a discrete number of packet streams in a layered format. The packet streams are hierarchical in the sense that an ordered subset of layers can be decoded to yield an output with fidelity proportional to the number of layers present, i.e., reconstruction quality improves (and bit rate increases) with each additional layer. Each layer provides refined quality in either the temporal or spatial domains, or potentially both.

PVH like most video compression algorithms exploits both the spatial and temporal redundancies inherent in natural video signals. Temporal redundancies are reduced through block-based "conditional replenishment". In this scheme, the video image is partitioned into a number of 16x16 blocks and only the blocks that change beyond some threshold are transmitted. Block updates can be striped across multiple layers to induce a temporal hierarchy. PVH does not use motion compensation or differential coding of pixel blocks.

PVH reduces spatial redundancy by compressing individual blocks and entropy-coding the output of a hybrid stage composed of a subband transform followed by a a Discrete Cosine Transform (DCT). The transform coefficients are progressively quantized to induce a layered representation.

A.2 The Coding Primitives

We first describe the signal source model and define a number of the primitives that we will exploit in the decoding algorithm.

A.2.1 Source Format

The video input signal is represented as a sequence of images sampled at an arbitrary rate up to the accuracy of the RTP timestamp. Synchronization and timing recovery are subsumed by RTP. The RTP media-specific timestamp for PVH is a simple 90kHz clock.

Each image is decomposed into an array of square pixels; the horizontal and vertical dimensions of each image must be a multiple of 16 pixels and in the range 16 to 4096. An image is represented in the YCbCr color space as a luminance component (Y) and two color difference components (Cb, Cr). CCIR Recommendation 601-1 specifies an 8-bit coding for each component where Y ranges over 0 to 219 with an offset of 16 while Cb and Cr range over -112 to +112 with an offset of 128. Hence, encoded values of Y fall in the absolute range $[16, 235]$ while Cb and Cr fall in $[16, 240]$. The YCbCr color space is related to the physical RGB space according to the following invertible transform:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.0 \\ 112.0 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where R, G, and B are each in $[0, 1]$.

The chrominance components are sampled at half the rate of the luminance component along both the horizontal and vertical dimensions of the signal. Each chrominance sample falls on the same spatial location as a luminance sample (i.e., this is usually referred to as 4:1:1 subsampling, compared to 4:2:0 subsampling where each chrominance pixel are centered over the corresponding 2x2 group of luminance pixels). The

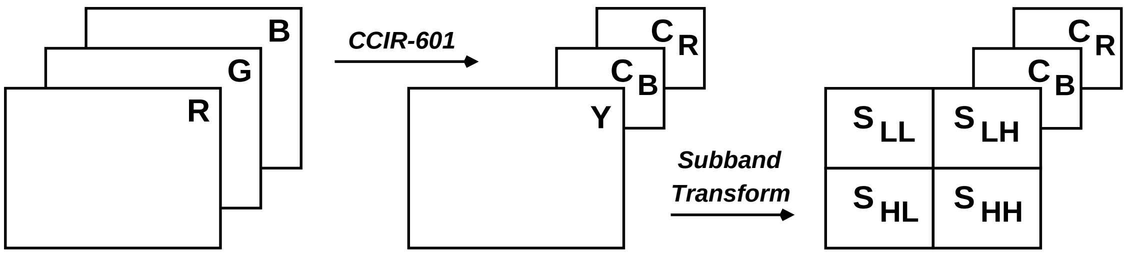


Figure A.1: **PVH Color Components.**

luminance component is transformed into four subbands according to the subband filtering operation below. Figure A.1 illustrates the six components that comprise each image. We call the subband filtered luminance components S_{LL}, S_{LH}, S_{HL}, and S_{HH}, and the two chrominance components Cb and Cr. In Version 1.0, S_{HH} is discarded.

A single-stage subband transform is applied to the entire luminance component to yield four subbands: S_{LL}, S_{LH}, S_{HL}, and S_{HH}. We use the following subband analysis filters:

$$\begin{aligned} H_0(z) &= (-1 + 3z^{-1} + 3z^{-2} - z^{-3})/4 \\ H_1(z) &= (-1 + 3z^{-1} - 3z^{-2} + z^{-3})/4 \end{aligned}$$

In both cases, we apply an even-symmetric extension of one point to the finite-length input signal at both ends. More specifically, call the input signal $x = (x[0], x[1], \ldots, x[N-1])$ of length N. We form a new signal $\hat{x} = (x[0], x[0], \ldots, x[N-1], x[N-1])$ of length $N+2$. The filtering process begins at lag 3 and ends at lag N+1 yielding exactly N output points. After subsampling by two (on even boundaries), the resulting signal has exactly $N/2$ points. (Note that N is necessarily even because the signal length is a multiple of 16.)

The input signal of dimension $W \times H$ is first filtered horizontally by H_0 and downsampled by two to yield a preliminary subband S_L of dimension $W/2 \times H$. Likewise, the input signal is filtered horizontally by H_1 and downsampled by two to yield a preliminary subband S_H. In turn, S_L is filtered by H_0 and downsampled by two vertically to yield S_{LL} of dimension $W/2 \times H/2$. Finally, filtering and dowsampling S_L with H_1 yields S_{LH}, S_H with H_0 yields S_{HL}, and S_H with H_1 yields S_{HH}.

A.2.2 Block Transform

PVH is a transform coding algorithm that exploits the Discrete Cosine Transform (DCT) over 8x8 blocks. The 8x8 DCT is defined as follows:

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

for $u, v, x, y \in [0, 7]$ and where

$$\begin{aligned} C(x) &= 1/\sqrt{2} \quad \text{for } x = 0 \\ &= 1 \quad\quad\ \ \text{o.w.} \end{aligned}$$

$f(\cdot)$ represents the image block signal in the pixel domain while $F(\cdot)$ represents the block of DCT transform coefficients. The pixel at coordinate $(0, 0)$ corresponds to the upper left hand corner of the block.

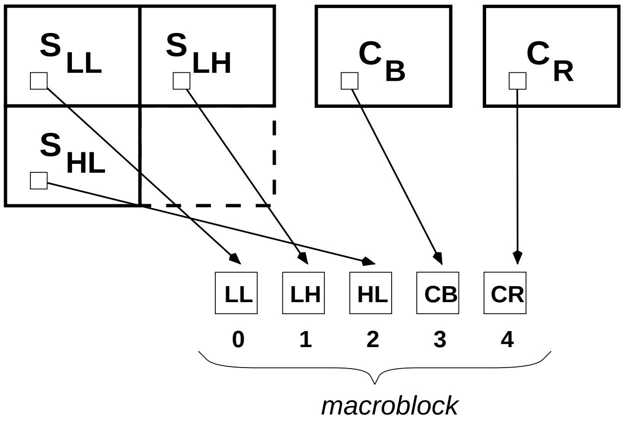


Figure A.2: **PVH Macroblock.**

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Table A.1: **DCT Scan Order**

A.2.3 Macroblock Structure

Although based on subband decomposition, PVH is a block-oriented compression algorithm that operates on 8x8 blocks. The 8x8 blocks are in turn aggregated into a 5-block unit called a *macroblock* as illustrated in Figure A.2. Here a block is taken from the same spatial location in each of the image components and aggregated to form the macroblock. Because of the downsampling operations carried out over the subbands as well as the chrominance components, a macroblock represents a 16x16 pixel block in the original signal. A DCT is applied to the LL, Cb, and Cr blocks while the LH and HL blocks are transformed no further.

The blocks are numbered as follows: (0) LL, (1) LH, (2) HL, (3) Cb, (4) Cr. We refer to these numbers as the component identifier (ID) of each block.

A.2.4 Coefficient Scan Order

DCT Scan Order. Because natural images tend to be dominated by low-frequency spectral content, the DCT coefficients in the upper left region of the block tend to contain most of the energy. Likewise, the lower right half of the block tends to have little energy and after quantization consists mostly of zeros.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Table A.2: **Subband Scan Order**

Thus effective zero run-length encoding can be carried out by ordering the coefficients into the so called *zig-zag* scan order. Whenever PVH processes all the DCT coefficients of a block, it scans them according to the order defined in Table A.1.

Subband Scan Order. Unlike the DCT, the subband transform does not concentrate energy in a particular region of a given band. Hence, we number the positions of subband coefficients within a bit-plane quad-tree using the simple left-to-right, top-to-bottom order shown in Table A.2. In this diagram, each numbered entry refers to the set of four spatially adjacent coefficients that we collectively call a subband coefficient quadruple or SBQ. Each coefficient within the SBQ is numbered as follows:

0	1
2	3

Whenever any one element of an SBQ is coded, the entire SBQ is coded jointly. For example, slot 1 in the subband scan order refers to coefficients 2, 3, 10, and 11, from the original 8x8 block.

A.2.5 Entropy Coder

Much of the PVH symbol stream is coded with a simple dictionary of symbols conditioned on the context of the bit parsing state. Many of these symbols are represented verbatim. In some cases, however, we exploit the zero-order entropy of the symbol stream to achieve higher compression gain by using variable-length Huffman codes. Although the Huffman codes used by PVH are non-adaptive, they can be externally specified. Dissemination of externally defined Huffman code tables as well as the selection of these tables is done out-of-band and is therefore outside the scope of this specification.

A.2.6 The Layering Model

The coded representation of the signal is explicitly arranged into an ordered set of *layers* that are individually mapped onto *channels*. Within each channel, the coded signal is formatted into discrete sets of bits or *packets*. A layer is a syntactically meaningful entity and the coded representation accounts explicitly for its structure. The decoder dynamically multiplexes bits from the different layers to parse and decode the coded signal. Channels, on the other hand, reflect no explicit syntax and are simply an abstract conduit by which layered data flows from the encoder to the decoder. For our purposes, a channel is merely an abstract label — in practice, it might be a network communication channel like an IP Multicast group or a disk I/O channel in a video server.

Although the layer number associated with each packet is needed to interpret the coded data, it does not explicitly appear in the data. Instead, it is inferred from the channel number, since the decoder can deduce the layer number from a channel number using control information in each packet. PVH assumes the channel number is communicated to the decoder as side information; it is likewise not encoded in band. The mapping from layers onto channels need not be static and more likely varies from frame to frame. Algorithms for dynamically adjusting the mapping in this fashion are beyond the scope of this specification.

Even if the encoder produces N layers, the decoder may consume only some number of layers $R < N$ to produce a reconstructed signal at a lower quality. In PVH, the subset of decodable layers must be contiguous, i.e., to decode R layers the decoder must have exactly layers 0 through $R-1$. Layer 0 is always required and is called the *base layer*.

The coded layers are carried from the encoder to the decoder by some means of channelized packet transport, where one or more layers may be dropped. Moreover, the transmission may be lossy and the decoder must gracefully tolerate packet erasures.

A.2.7 Progressive Quantization

The AC coefficients from the DCT transform and the LH and HL bands of the subband transformations are both progressively quantized. By distributing progressive refinements across layers, we create a representation where spatial quality improves as we process additional layers. Each spatial layer in the coded stream provides progressive refinement of some subset of these coefficients. The DC coefficient from the DCT is quantized with a simple uniform quantizer and is coded in the base layer.

Before progressive refinement, the AC DCT coefficients are quantized by the uniform, mid-step base-layer quantizer 2^{N_D} with a dead-zone twice the step size and symmetric about zero. Likewise, the subband coefficients are quantized by the uniform, mid-step quantizer 2^{N_S}. The quantization function $Q(\cdot)$ is given by

$$\begin{aligned} Q_N(x) &= \lceil x/2^N \rceil \quad x < 0 \\ &= \lfloor x/2^N \rfloor \quad x \geq 0 \end{aligned}$$

If C_D is an AC DCT coefficient and $\hat{C}_D$ is its quantized value then $\hat{C}_D = Q_{N_D}(C_D)$. Likewise if C_S is an subband coefficient and $\hat{C}_S$ is its quantized value then $\hat{C}_S = Q_{N_S}(C_S)$.

Once the DCT and subband coefficients are quantized and coded onto their assigned layers, subsequent layers progressively refine the initial quantization intervals by supplying additional information for each coefficient. On each pass, the quantization step is exactly halved by sending an extra bit of precision of the magnitude of each transform coefficient.

The method by which an additional bit of precision is represented is different for the DCT and the subband coefficients. For the DCT, we refine each coefficient by jointly coding the entire bit-plane of the next significant bit position of the AC coefficient magnitudes in a fashion analogous to progressive-mode JPEG [85, Annex G]. A refinement bit-plane is coded by first refining each previously transmitted coefficient magnitude by one bit and then identifying coefficient positions that become significant on this pass using a series of run-lengths over the zig-zag scan order. For the subband coefficients, an additional bit of precision is encoded with a "quad-tree" representation of the bit-plane. In this structure, the bit-plane is recursively subdivided into four equal sized subregions until each non-zero bit is identified.

In both cases, after the last refinement, each magnitude is biased by one-half of the final quantization step size to center the reconstructed value at mid-step in the quantization interval.

The DC DCT coefficient is not progressively refined. Instead it is encoded at full precision in the base layer. An externally defined profile indicates whether the DC coefficient is coded directly or is spatially predicted from the previous macroblock. (Version 1.0 does not support spatial prediction mode.)

The AC DCT coefficients can be weighted to reflect the relative perceptual importance of the different coefficients by pre-multiplying each coefficient with an entry in a weighting matrix. The decoder multiplies by the inverse weight to reconstruct the original (but quantized) value. (Version 1.0 does not support this feature.)

A.2.8 Block Addressing

The input image is partitioned into an integer number of 16x16 blocks that cover the entire image. The blocks are addressed in normal "scan order". Block 0 is in the upper left corner and the last block is in the lower right corner. The blocks are sequentially number first from left to right then from top to bottom. Due to the subband decomposition process and the chrominance downsampling operation, each macroblock is composed of a number of 8x8 rather than 16x16 blocks each from a different component. The five 8x8 blocks (LL, LH, HL, Cb, and Cr) of the same macroblock are each spatially centered over the identical 16x16 pixel block from the original image.

A.3 Structure of the Coded Bit Sequence

The coded bit sequence represents an unbounded sequence of images sampled at arbitrary points in time. We use a hierarchical decomposition to describe the image syntax at a number of levels:

- the image level,
- the slice level,
- the macroblock level, and
- the block level.

A.3.1 Image Level

Unlike traditional video coding schemes, the image level semantics do not explicitly appear in the compressed symbol stream. Instead, image boundaries, coding synchronization points, and timing recovery are all subsumed by the RTP packet framing protocol. Syntactically, each image is partitioned into a set of *slices* as in MPEG. The slice boundaries must be contiguous, but the last slice of an image may be omitted (the end of an image is marked by RTP framing information).

A.3.2 Slice Level

A *slice* is the fundamental coding unit. Each slice is arranged into a discrete set of layers and there is exactly one coded symbol string per layer. For purposes of exposition, we refer to these symbol strings as L_0 through L_{N-1}.

A slice is an idempotent data item that is both spatially and temporally independent of all other slices. It consists of a variable number of contiguous macroblocks all within the same image. Within a slice, macroblocks may be omitted by skip codes embedded in the symbol stream, but the slice logically covers all macroblocks from and up to adjacent slices.

A.3.3 Macroblock Level

A macroblock consists of a bit-allocation descriptor (BD) followed by the coded blocks. The BD describes which block types are present, where the base quantization of each layer falls, and how each block is individually coded onto layers (i.e., how the refinement passes are spread across the layers $L_0 \ldots L_{N-1}$). The BD for all three component types is coded as a unit rather than individually. The BD is predicted across macroblocks. When it does not change, it is coded in a single bit.

The BD defines the bit allocation fingerprint for three block types: luminance DCT (LUM-DCT), luminance subband (LUM-SBC), and chrominance DCT (CHM). The LL block is type LUM-DCT, the LH and HL blocks are type LUM-SBC, and the Cb and Cr blocks are type CHM.

The five blocks that comprise the macroblock are encoded sequentially in increasing order of their component ID.

A.3.4 Block Level

Each block of a macroblock is coded onto the output layers according to the previously defined BD for its block type. The output symbols are a function of the signal data and the BD quantization layout.

A.4 The Packet Framing Protocol

The syntactic structures in the previous section are mapped onto a collection of packets each of which is assigned in its entirety to a specific layer. An image is defined by a collection of packets from potentially different layers. Each packet is prefixed by a standard RTP header as described in [153]. The RTP payload type is not pre-assigned and is instead dynamically assigned by an external mechanism (e.g., using the Session Description Protocol (SDP) [75]).

All packets that belong to an identical image have the same RTP timestamp but are in no particular order. Moreover, the semantics of the RTP timestamp field are the same across each layer and are derived from the same time base. In particular, the random offset added to each timestamp (as required by RTP) is consistent across each layer. It is recommended (but not required) that all packets from a given layer are generated in increasing order of the block addresses of the blocks contained in the packet. The decoder must be able to process packets received out of order.

The RTP Marker Bit is set on the last packet of each layer for a particular image; otherwise it is clear. Note that if a packet containing a set marker bit is lost, a subsequent timestamp change explicitly indicates that the previous image is terminated (at the expense of additional delay). The last packet of an image need not contain the last block of the image, i.e., the marker bit can terminate the image prematurely.

Although an image is comprised of a set of packets, PVH imposes a "group structure" on the packets that make up an image. That is, each image consists of a number of "packet groups" that correspond one-to-one with slices. Each slice is rooted in exactly one base-layer packet, which explicitly defines the range of macroblocks for that slice. All packets at higher layers that overlap this range are thus members of that same slice. Note that any non-base-layer packet can straddle slice boundaries; a packet need not be contained in a single slice.

Within a slice, bits are distributed across packets that correspond to different layers (but are presented to the decoder on different channels). Typically multiple packets are needed at higher layers to contain the entire slice and thus the slice must be fragmented across multiple packets. In this case, each packet begins and ends on a macroblock boundary and the packet header explicitly indicates the range of macroblocks

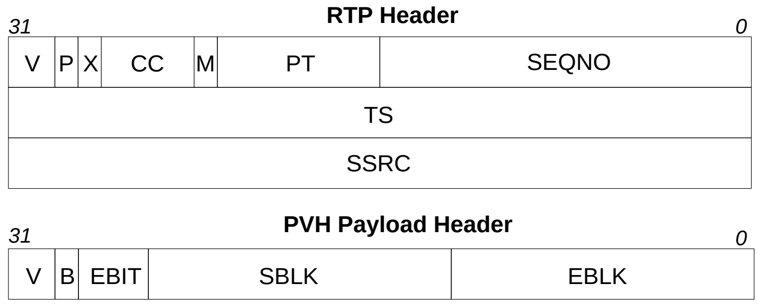


Figure A.3: **RTP/PVH Packet Headers.**

field	*width*	
V	2	The version number of the codec (currently 0).
B	1	A base-layer indicator set when the packet represents base-layer information.
EBIT	3	The number of bits in the last octet of the packet to ignore.
SBLK	13	The block address of the first coded block in the packet.
EBLK	13	The block address of the last coded block in the packet.

Table A.3: **Fixed Packet Header Fields**

present. This subset of macroblocks from the original slice is referred to as a *sub-slice*. When a packet straddles a slice boundary, header information in lower-layer packets explicitly indicates the offset where each slice begins.

A.4.1 Slice Reassembly

Given that slices are spread across multiple packets, the decoder must reassemble the packets into the codewords $L_0 \ldots L_{N-1}$ that comprise the slice. Once we have the codeword for the slice, we can decode the set of macroblocks in the slice using the codec syntax. We recover the slice according to the group structure defined above using header information in the packet stream, in addition to the standard header fields provided by RTP.

The PVH payload format header immediately follows the RTP header in each packet. The structure of the PVH header is illustrated in Figure A.3 and the semantics of each field are briefly summarized in Table A.3. The first field in the header is a version number to allow for incremental deployment of new versions of the codec. The base-layer bit B marks each base-layer packet. Without this explicit indication, the decoder cannot determine where the base-layer begins (in the face of packet loss, delay, and reordering). The EBIT field indicates the number of bits from the last octet to omit since typical communication and storage interfaces assume a byte rather than bit granularity. Finally, the SBLK and EBLK fields specify the macroblock address ranges of the blocks coded within the packet. If B is set, then SBLK and EBLK define a slice; otherwise, SBLK and EBLK define a sub-slice.

In packets with B set, a number of additional fields, described in Table A.4, immediately follow the PVH header. The width and height appear first and indicate the geometry of the underlying image sequence.

field	*width*	
W	8	Image width. One less than the image width divided by 16.
H	8	Image height. One less than the image width divided by 16.
NR	4	Number of resumption layer/pointer pairs.
RL(1)	4	Resumption layer 1.
RP(1)	16	Resumption pointer 1.
...	...	...
RL(NR)	4	Resumption layer NR.
RP(NR)	16	Resumption pointer NR.

Table A.4: **Base Layer Fixed Fields**

Following the width and height is a list of resumption layers and pointers to facilitate slice reassembly. This list defines both a horizontal relationship among syntactic layers (i.e., how offsets into packets for bit parsing can be aligned) as well as a vertical relationship (i.e., where to find the next higher layer within the set of channels). The resumption layer specifies the channel on which the next layer is found (i.e., channels need not be contiguous), while the resumption pointer gives bit offset into the packet in the next significant syntactic layer that is syntactically aligned with the start of this packet. The channel numbers of each layer above the base layer are explicitly listed as the 4-bit resumption layers. If the address of the first block in the base-layer packet is A, then for each enhancement layer, the 13-bit resumption pointer gives the bit offset into the enhancement packet where block A is coded.

The codeword string L_k is formed by concatenating the contents of the sub-slices from layer k using the resumption offsets and pointers in the base layer. L_0 is simply the contents of the single base-layer packet and requires no concatenation.

Assuming all packets that comprise a slice are present, fully formed codewords can be extracted from the packets by concatenating adjacent sub-slices across each layer. Since the macroblock boundaries of each slice are adjacent, the concatenation order is explicit in the packet header. Although the bit stream and resumption pointers are specifically designed to recover gracefully from packet loss, algorithms to do so are beyond the scope of this specification.

A.5 The Slice Coded Bit Sequence

This section specifies the precise decoding algorithm for the coded symbol strings $L_0 \dots L_{N-1}$ obtained as above that together form a slice. Each macroblock is decoded in exactly the same fashion. Hence, it suffices to describe the algorithm for one macroblock.

The first codeword in the slice appears in L_0 and identifies the initial bit-allocation descriptor (BD).

A.5.1 Default BD

The default bit-allocation descriptors are initially decoded from L_0. There is a separate descriptor for each of the three block types. Each descriptor defines the layer on which the first (i.e., base layer) pass falls, and the number of refinement passes that appear in subsequent layers including the base layer (a refinement pass may be included in the same layer as the base-layer pass). In the remainder of this document

symbol	*codeword*
1	1
2-17	10xxxx
≥ 18	00xxxxxxxxxxx

Table A.5: **Macroblock Addressing Codes**

$B(k)$ refers to the channel on which the base-layer pass for block type k appears, $Q(k)$ refers to the current base-layer quantizer for block type k, and $R_m(k)$ refers to the number of refinement passes at channel m for block type k.

The descriptors for each of the block types are coded in sequence:

D_0	D_1	D_2

The descriptor D_k is decoded as follows. The next bit is parsed from L_0: if it is a 1, then $B(k)$ is the channel number on which the base-layer packet arrived. Otherwise, bits are read from L_0 until a 1 is encountered. Call the number of zeros read Z. Then $B(k)$ is the channel number indicated by the Zth resumption pointer in the base-layer packet. A three-bit word is next read from L_0 and this word is the binary representation of $Q(k)$. $NL - B(k)$ three-bit words are finally read from L_0 and these values respectively represent $R_{B(k)}(k), \ldots, R_{NL}(k)$. Each $\{R_l(k) \text{ for } 0 \leq l < B(k)\}$ is undefined.

A.5.2 Macroblock Addressing

The macroblock address is the only state other than the BD that is predicted across block boundaries. Call the current address MBA. We initialize MBA with the start address of the slice (given by the SBLK field in the base-layer packet) and decode subsequent MBA's using differential prediction to skip over unreplenished blocks.

Except for the first block, the MBA is decoded at the start of each block by parsing the next Huffman code from L_0 using Table A.5. The values 2-17 are coded in the 4-bit binary representation of their value biased by -2 and prefixed with 10. Values larger than 17 are escaped with 00 and coded verbatim in 11 bits (with no bias).

A.5.3 Macroblock Layer

Following the macroblock address is the macroblock data. Each macroblock is optionally prefixed by a local BD. The next bit from L_0 is read: if 0, then there is no BD; otherwise, a BD is decoded from L_0 as in § A.5.1. This BD remains in use until overridden by a local BD definition in some future block.

Following the BD is the coded symbol stream for each block appearing in the following order:

LL	LH	HL	Cb	Cr

The LL, Cb, and Cr blocks of DCT coefficients are decoded with the DCT process in the next section, and the LH and HL 8x8 blocks of subband coefficients are decoded with the subband coefficient process in the subsequent section. The descriptor D_0 is used for LL, D_1 is used for LH and HL, and D_2 is used for Cb and Cr. Unless terminated by the end of the slice, the next macroblock address code immediately follows the Cr block data.

A.5.4 DCT Coefficient Decoding

The DCT decoding process is carried out in a number of stages. In the first stage, we decode a set of initial base-layer coefficients by parsing bits from $L_{B(k)}$ (for $k = 0$ or 2).

DCT Base Layer

The base-layer coefficients are decoded in a fashion similar to point transform technique in progressive-mode JPEG. PVH uses a uniform mid-step quantizer $2^{Q(k)}$ applied to each AC coefficient.

The next 8 bits of $L_{B(k)}$ are read and multiplied by 8 to form the DC DCT coefficient. (The DC coefficient has no quantization bias, i.e., the encoder uses rounding rather than truncation to quantize the DC level.) The AC coefficients are then decoded as follows:

(1) Initialize the variable OFFSET to 0.

(2) Parse the next Huffman code from $L_{B(k)}$ using Table A.6.

(3) If this symbol is EOB, the block is finished.

(4) If this symbol is ESC, then read the next 6 bits to form the run-length, the subsequent 8 bits to form the level (in sign-magnitude representation).

(5) Scale the level by $2^{Q(k)}$ to recover the pre-quantized coefficient level.

(6) Add the run-length to OFFSET. The decoded level from (5) then corresponds to the DCT coefficient at location OFFSET in zig-zag scan order. Resume decoding at step (2).

Upon reaching an EOB symbol, which must terminate the block, all coefficients not encountered are assumed to be 0. The set of non-zero, decoded coefficients must be maintained for later passes because it provides conditioning context for refinement.

Once the initial coefficients have been decoded, zero or more enhancement passes may follow on the same and/or higher layers according to D_0 and D_2. Each enhancement pass provides exactly one additional bit of precision by effectively coding the next bit-plane of DCT coefficients. An enhancement bit plane is coded in two passes. In the first pass, an additional bit of precision of each non-zero coefficient is encoded and in the second pass, coefficients that become non-zero at the given bit position are identified. If the current layer being processed is m, then the two-stage process repeats itself $R_m(k)$ times reading symbols from L_m.

DCT Refinement

In the refinement pass, each non-zero coefficient is refined by exactly one bit. If there are N non-zero coefficients and the current layer being processed is m, then the next N bits are read from L_m and each bit individually refines a coefficient magnitude in zig-zag scan order.

DCT Identification

In the identification pass, coefficients that become non-zero at the current bit position are identified. The process is as follows:

(1) Initialize OFFSET to 0.

run	*level*	*codeword*	*run*	*level*	*codeword*	*run*	*level*	*codeword*
EOB	-	10	ESC	-	000001			
0	1	110	1	7	00000000101010	8	2	0000000100010
0	-1	111	1	-7	00000000101011	8	-2	0000000100011
0	2	01000	2	1	01010	9	1	00001010
0	-2	01001	2	-1	01011	9	-1	00001011
0	3	001010	2	2	00001000	9	2	00000000100010
0	-3	001011	2	-2	00001001	9	-2	00000000100011
0	4	00001100	2	3	00000010110	10	1	001001110
0	-4	00001101	2	-3	00000010111	10	-1	001001111
0	5	001001100	2	4	0000000101000	10	2	00000000100000
0	-5	001001101	2	-4	0000000101001	10	-2	00000000100001
0	6	001000010	2	5	00000000101000	11	1	001000110
0	-6	001000011	2	-5	00000000101001	11	-1	001000111
0	7	00000010100	3	1	001110	12	1	001000100
0	-7	00000010101	3	-1	001111	12	-1	001000101
0	8	0000000111010	3	2	001001000	13	1	001000000
0	-8	0000000111011	3	-2	001001001	13	-1	001000001
0	9	0000000110000	3	3	0000000111000	14	1	00000011100
0	-9	0000000110001	3	-3	0000000111001	14	-1	00000011101
0	10	0000000100110	3	4	00000000100110	15	1	00000011010
0	-10	0000000100111	3	-4	00000000100111	15	-1	00000011011
0	11	0000000100000	4	1	001100	16	1	00000010000
0	-11	0000000100001	4	-1	001101	16	-1	00000010001
0	12	00000000110100	4	2	00000011110	17	1	0000000111110
0	-12	00000000110101	4	-2	00000011111	17	-1	0000000111111
0	13	00000000110010	4	3	0000000100100	18	1	0000000110100
0	-13	00000000110011	4	-3	0000000100101	18	-1	0000000110101
0	14	00000000110000	5	1	0001110	19	1	0000000110010
0	-14	00000000110001	5	-1	0001111	19	-1	0000000110011
0	15	00000000101110	5	2	00000010010	20	1	0000000101110
0	-15	00000000101111	5	-2	00000010011	20	-1	0000000101111
1	1	0110	5	3	00000000100100	21	1	0000000101100
1	-1	0111	5	-3	00000000100101	21	-1	0000000101101
1	2	0001100	6	1	0001010	22	1	00000000111110
1	-2	0001101	6	-1	0001011	22	-1	00000000111111
1	3	001001010	6	2	0000000111100	23	1	00000000111100
1	-3	001001011	6	-2	0000000111101	23	-1	00000000111101
1	4	00000011000	7	1	0001000	24	1	00000000111010
1	-4	00000011001	7	-1	0001001	24	-1	00000000111011
1	5	0000000110110	7	2	0000000101010	25	1	00000000111000
1	-5	0000000110111	7	-2	0000000101011	25	-1	00000000111001
1	6	00000000101100	8	1	00001110	26	1	00000000110110
1	-6	00000000101101	8	-1	00001111	26	-1	00000000110111

Table A.6: **DCT Base-layer Coefficient/Run Codes**

symbol	*codeword*	*symbol*	*codeword*
EOB	101	5	0011
1	11	6	01110
2	100	7	01100
3	010	8	00100
4	000	ESC	01111

Table A.7: **DCT Run-length Codes**

(2) Decode the next Huffman code from L_m using Table A.7.

(3) If EOB, this pass is complete; end it.

(4) If ESCAPE, read the next 6 bits from L_m to form the run-length.

(5) Skip past "run-length" zero-valued coefficients starting from the coefficient at position OFFSET in zig-zag order. Call this position NEW_OFFSET.

(6) Set the magnitude of the coefficient at position NEW_OFFSET in zig-zag order to 2^n where n is the current refinement bit position.

(7) Set OFFSET to NEW_OFFSET. Resume decoding at step (2).

After the final refinement pass, each AC coefficient is biased to the mid-range of the final quantization step size. For example, if the last refined bit position is bit 3 (i.e., 2^3), then 4 is added to each non-zero coefficient magnitude.

A.5.5 Subband Coefficient Decoding

The subband coefficient decoding process is carried out in a number of stages. In the first stage, we decode a set of initial base-layer coefficients by parsing bits from $L_{B(1)}$ and in subsequent stages we decode bit-planes of subband coefficients to incrementally enhance the reconstructed signal.

Quad-tree Code

In both the base-layer and refinement passes, we use a quad-tree code to identify which SBQ's are literally encoded and which are zero and therefore suppressed. Since there are 16 SBQ's in a block, the quad-tree symbol is a 16-element bit vector. If an SBQ is present, then its scan position is indicated by setting the corresponding bit in the bit vector; otherwise, the bit is clear.

Typically the quad-tree descriptor has many zeros and is compactly represented with the following variable length recursive codeword:

- If the first bit is 0, the entire descriptor is 0.
- Otherwise, we inspect the next 4 bits. Each of these bits corresponds to a 4-tuple in the 16-element descriptor, appearing in left to right order.
  - If a bit is 0, then the corresponding 4-tuple is all zeros.
  - Otherwise, we read the next 4 bits from the codeword to form the corresponding 4-tuple.

symbol	*codeword*	*symbol*	*codeword*
0	0		
1	100	-1	101
2	110010	-2	111010
3	110011	-3	111011
4	110100	-4	111100
5	110101	-5	111101
6	110110	-6	111110
7	110111	-7	111111

Table A.8: **Subband Coefficient Codes**

Subband Base Layer

The base layer is decoded by first parsing a quad-tree descriptor from $L_{B(1)}$. For each SBQ indicated in the quad-tree descriptor, we decode the four coefficients of the SBQ using the Huffman code in Table A.8. The magnitude of each reconstructed level is scaled by $Q(1)$ to recover the original coefficient value. For each SBQ not present, we set the corresponding entries to 0.

In this default Huffman table, the largest coefficient magnitude is 7. Hence, the default subband base-layer quantizer must be large enough to guarantee no more than 3 bits of precision for the base-layer pass. Arbitrary precision is realizable through refinement passes.

Subband Refinement Pass

Unlike the DCT refinement process which consists of two passes, the subband refinement step is carried out in a single pass. To decode the bit plane of refinement from layer m, the decoder executes the following algorithm:

- Read a quad-tree descriptor from L_m.
- For each SBQ indicated by the descriptor in subband scan order:
  - Read one bit from L_m and refine the corresponding coefficient magnitude.
  - If the coefficient was previously 0, read one more bit. If this bit is 1, then the newly significant coefficient is negative; otherwise, it is positive.

This process is repeated for $R_m(k)$ iterations, each time at the next less significant bit position.

Bibliography

[1] E. H. Adelson, E. Simoncelli, and R. Hingorani. Orthogonal pyramid transforms for image coding. In *Proc. SPIE*, volume 845, pages 50–58, Cambridge, MA, October 1987.

[2] Andres Albanese, Johannes Blömer, Jeff Edmonds, and Michael Luby. Priority encoding transmission. Technical Report TR-94-039, International Computer Science Institute, Berkeley, CA, September 1994.

[3] Elan Amir, Steven McCanne, and Martin Vetterli. A layered DCT coder for Internet video. In *Proceedings of the IEEE International Conference on Image Processing*, pages 13–16, Lausanne, Switzerland, September 1996.

[4] Elan Amir, Steven McCanne, and Hui Zhang. An application-level video gateway. In *Proceedings of ACM Multimedia '95*, pages 255–265, San Francisco, CA, November 1995. ACM.

[5] T. Bailly, Bernard Gold, and Stephanie Seneff. A technique for adaptive voice flow control in integrated packet networks. *IEEE Transactions on Communications*, COM-28(4):325–333, March 1980.

[6] D. Balenson. *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers*. ARPANET Working Group Requests for Comment, DDN Network Information Center, February 1993. RFC-1423.

[7] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT) an architecture for scalable inter-domain multicast routing. In *Proceedings of SIGCOMM '93*, pages 85–95, San Francisco, CA, September 1993. ACM.

[8] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang. The Tenet real-time protocol suite: Design, implementation, and experiences. *IEEE/ACM Transactions on Networking*, 4(1):1–10, February 1996.

[9] A. Banerjea, E. Knightly, F. Templin, and H. Zhang. Experiments with the Tenet real-time protocol suite on the Sequoia 2000 wide area network. In *Proceedings of ACM Multimedia '94*, San Francisco, CA, October 1994.

[10] L. Berc, W. Fenner, R. Frederick, and S. McCanne. *RTP Payload Format for JPEG-compressed Video*. Internet Engineering Task Force, Audio-Video Transport Working Group, November 1995. Internet Draft (work in progress).

[11] Arthur W. Berger, Samuel P. Morgan, and Amy R. Reibman. Statistical multiplexing of layered video streams over ATM networks with leaky-bucket traffic descriptors. *Submitted to IEEE Transactions on Networking*, 1993.

[12] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielson, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[13] Steven Langley Blake. *Optimized Two-Layer DCT-Based Video Compression Algorithm for Packet-switched Network Transmission*. PhD thesis, North Carolina State University, 1995.

[14] Jean-Chrysostome Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of SIGCOMM '93*, pages 289–298, San Francisco, CA, September 1993. ACM.

[15] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. Scalable feedback control for multicast video distribution in the Internet. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994. ACM.

[16] C. Mic Borman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28:119–125, 1995.

[17] R. Braden, L. Zhang, D. Estrin, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 function specification, November 1996. Internet Draft (RFC pending).

[18] Tom Brown, Sharif Sazzad, Charles Schroeder, Pierce Cantrell, and Jerry Gibson. Packet video for heterogeneous networks using CU-SeeMe. In *Proceedings of the IEEE International Conference on Image Processing*, pages 9–12, Lausanne, Switzerland, September 1996.

[19] Peter J. Burt and Edward H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, COM-31(4):532–540, April 1983.

[20] Ingo Busse, Bernd Deffner, and Henning Schulzrinne. Dynamic QoS control of multimedia applications based on RTP. In *Proceedings of the First International Workshop on High Speed Networks and Open Distributed Platforms*, St. Petersburg, Russia, June 1995.

[21] A. Campbell and G. Coulson. A QoS adaptive transport system: Design, implementation and experience. In *Proceedings of ACM Multimedia '96*, Boston, MA, November 1996. ACM.

[22] S. Casner, J. Lynn, P. Park, K. Schroder, and C. Topolcic. *Experimental Internet Stream Protocol, version 2 (ST-II)*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, October 1990. RFC-1190.

[23] Steve Casner and Steve Deering. First IETF Internet audiocast. *ConneXions*, 6(6):10–17, 1992.

[24] Navin Chaddha. Software only scalable video delivery system for multimedia applications over hetrogeneous networks. In *Proceedings of the IEEE International Conference on Image Processing*, Washington, DC, October 1995.

[25] Navin Chaddha and Anoop Gupta. A frame-work for live multicast of video streams over the Internet. In *Proceedings of the IEEE International Conference on Image Processing*, pages 1–4, Lausanne, Switzerland, September 1996.

[26] Navin Chaddha, Gerard A. Wall, and Brian Schmidt. An end to end software only scalable video delivery system. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video*, Durham, NH, April 1995. ACM.

[27] Shun Yan Cheung, Mostafa H. Ammar, and Xue Li. On the use of destination set grouping to improve fairness in multicast video distribution. In *Proceedings IEEE Infocom '96*, pages 553–560, San Francisco, CA, March 1996.

[28] Hin Soon Chin, John W. Goodge, Roy Griffiths, and David J. Parish. Statistics of video signals for viewphone-type pictures. *IEEE Journal on Selected Areas in Communications*, 7(5):826–832, June 1989.

[29] C. A. Christopoulos, A. N. Skodras, and J. Cornelis. Comparative performance evaluation of algorithms for fast computation of the two-dimensional DCT. In *Proceedings of the IEFFF Benelux and ProRISC Workshop on Circuits, Systems and Signal Processing*, Papendal, Arnhen, March 1994.

[30] David Clark, Scott Shenker, and Lixia Zhang. Supporting realtime applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of SIGCOMM '92*, pages 14–26, Baltimore, Maryland, August 1992. ACM.

[31] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of SIGCOMM '90*, Philadelphia, PA, September 1990. ACM.

[32] D. Cohen. On packet speech communication. In *Proceedings of the Fifth International Conference on Computer Communications*, pages 271–274, Atlanta, Georgia, October 1980. IEEE.

[33] Danny Cohen. *Specifications for the Network Voice Protocol (NVP).* ARPANET Working Group Requests for Comment, DDN Network Information Center, Information Sciences Institute, 1976. RFC-741.

[34] Charles Compton and David Tennenhouse. Collaborative load shedding for media-based applications. *International Conference on Multimedia Computing and Systems*, May 1994.

[35] Thomas M. Cover. Broadcast channels. *IEEE Transactions on Information Theory*, IT-18(1):2–14, January 1972.

[36] Earl Craighill, Martin Fong, Keith Skinner, Ruth Lang, and Kathryn Gruenefeldt. SCOOT: An object-oriented toolkit for multimedia collaboration. In *Proceedings of ACM Multimedia '94*, pages 41–49. ACM, October 1994.

[37] John M. Danskin, Geoffrey M. Davis, and Xiyong Song. Fast lossy Internet image transmission. In *Proceedings of ACM Multimedia '95*, pages 321–332, San Francisco, CA, November 1995. ACM.

[38] Peter B. Danzig. *Optimally Selecting the Parameters of Adaptive Backoff Algorithms for Computer Networks and Multiprocessors*. PhD thesis, University of California, Berkeley, December 1989.

[39] John C. Darragh and Richard L. Baker. Fixed distortion subband coding of images for packet-switched networks. *IEEE Journal on Selected Areas in Communications*, 7(5):789–800, June 1989.

[40] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An architecture for wide-area multicast routing. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994. ACM.

[41] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[42] Stephen E. Deering and Robert M. Hinden. *Internet Protocol, Verion 6 (IPv6)*. Internet Engineering Task Force, IPNG Working Group, December 1995. RFC-1883.

[43] Steve Deering. Internet multicast routing: State of the art and open research issues, October 1993. Multimedia Integrated Conferencing for Europe (MICE) Seminar at the Swedish Institute of Computer Science, Stockholm.

[44] Luca Delgrossi, Christian Halstrick, Dietmar Hehmann, Ralf Guido Herrtwich, Oliver Krone, Jochen Sandvoss, and Carsten Vogt. Media scaling for audiovisual communication with the Heidelberg transport system. In *Proceedings of ACM Multimedia '93*, pages 99–104. ACM, August 1993.

[45] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1:3–26, 1990.

[46] Tim Dorcey. CU-SeeMe desktop videoconferencing software. *ConneXions*, 9(3), March 1995.

[47] Tim Dorcey and Richard Cogger. *CU-SeeMe*. Cornell University. Software on-line[1].

[48] Ernest A. Edmonds, Linda Candy, Rachel Jones, and Bassel Soufi. Support for collaborative design: Agents and emergece. *Communications of the ACM*, 37(7):41–47, July 1994.

[49] Alexandros Eleftheriadis, Sassan Pejhan, and Dimitris Anastassiou. Algorithms and performance evaluation of the Xphone multimedia communication system. In *Proceedings of ACM Multimedia '93*, pages 311–320. ACM, August 1993.

[50] William H. R. Equitz and Thomas M. Cover. Successive refinement of information. *IEEE Transactions on Information Theory*, 37(2):269–275, March 1991.

[51] Kevin Fall, Joseph Pasquale, and Steven McCanne. Workstation video playback performance with competitive process load. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video*, pages 179–182, Durham, NH, April 1995.

[52] W. Fenner. *Internet Group Management Protocol, Version 2*. Internet Engineering Task Force, Inter-Domain Multicast Routing Working Group, February 1996. Internet Draft (work in progress).

[53] Domenico Ferrari and Dinesh Verma. A scheme for real-time communication services in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[54] Sally Floyd. Notes on CBQ and guaranteed service. Unpublished note available in ftp://ftp.ee.lbl.gov/papers/guaranteed.ps, July 1995.

[55] Sally Floyd and Van Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[56] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[57] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. In *Proceedings of SIGCOMM '93*, pages 33–44, San Francisco, CA, September 1993. ACM.

[1] ftp://gated.cornell.edu/pub/video

[58] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[59] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of SIGCOMM '95*, pages 342–356, Boston, MA, September 1995. ACM.

[60] Armando Fox, Steven D. Gribble, and Eric A. Brewer Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of ASPLOS '96*, Cambridge, MA, October 1996.

[61] Ron Frederick. *Network Video (nv)*. Xerox Palo Alto Research Center. Software on-line[2].

[62] Ron Frederick. Experiences with real-time software video compression. In *Proceedings of the Sixth International Workshop on Packet Video*, Portland, OR, September 1994.

[63] Mark W. Garrett and Martin Vetterli. Congestion control strategies for packet video. In *Proceedings of the Fourth International Workshop on Packet Video*, pages 1–6, Kyoto, Japan, March 1991.

[64] Mark W. Garrett and Martin Vetterli. Joint source/channel coding of statistically multiplexed real-time services on packet networks. *IEEE/ACM Transactions on Networking*, 1(1):71–80, February 1993.

[65] Mark W. Garrett and Walter Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994. ACM.

[66] Mark William Garrett. *Contributions Toward Real-Time Services on Packet Switched Networks*. PhD thesis, Columbia University, 1993.

[67] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991.

[68] M. Ghanbari. Two-layer coding of video signals for VBR networks. *IEEE Journal on Selected Areas in Communications*, 7(5):771–781, June 1989.

[69] H. Gharavi. Subband coding of video signals. In J. W. Woods, editor, *Subband Image Coding*. Kluwer Academic Publishers, Boston, MA, 1990.

[70] M. Gilge and R. Gusella. Motion video coding for packet-switching networks—an integrated approach. In *Proceedings of the SPIE Conference on Visual Communications and Image Processing*, Boston, MA, November 1991. ACM.

[71] Amit Gupta, Wendy Heffner, Mark Moran, and Clemens Szyperski. Network support for realtime multi-party applications. In *Proceedings of the Fourth International Workshop on Network and OS Support for Digital Audio and Video*, Lancaster, U.K., November 1993. ACM.

[72] A. Haar. Zur Theorie der orthogonalen Funktionensysteme. *Math. Annal.*, 69:331–371, 1910.

[2] ftp://ftp.parc.xerox.com/net-research

[73] Mark Handley. *Session DiRectory*. University College London. Software on-line[3].

[74] Mark Handley and Jon Crowcroft. Internet multimedia conferencing architecture. *ConneXions*, 10(6):2–13, June 1996.

[75] Mark Handley and Van Jacobson. SDP: Session description protocol, November 1995. Internet Draft (work in progress).

[76] Mark Handley and Ian Wakeman. CCCP: Conference control channel protocol, a scalable base for building conference control applications. In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995. ACM.

[77] Mark J. Handley. Using the UCL H.261 codec controller, December 1993. On-line html document[4].

[78] Paul Haskell and David Messerschmitt. Open network architecture for continuous-media services: The medley gateway. Technical report, University of California, Berkeley, CA, January 1994.

[79] Paul Heckbert. Color image quantization for frame buffer display. In *Proceedings of SIGGRAPH '82*, page 297, 1982.

[80] Wendy Heffner. Scaling issues in the design and implementation of the Tenet RCAP2 signaling protocol. Technical Report TR-95-022, International Computer Science Institute, Berkeley, CA, May 1995.

[81] Andrew T. Heybey. Video coding and the application level framing protocol architecture. Technical Report MITE/LCS/TR-542, Massachusetts Institute of Technology, Cambridge, MA, June 1992.

[82] Don Hoffman, Gerard Fernando, and Vivek Goyal. *RTP Payload Format for MPEG1/MPEG2 Video*. Internet Engineering Task Force, Audio-Video Transport Working Group, June 1995. Internet Draft (work in progress).

[83] Don Hoffman and Michael Speer. Hierarchical video distribution over Internet-style networks. In *Proceedings of the IEEE International Conference on Image Processing*, pages 5–8, Lausanne, Switzerland, September 1996.

[84] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of IRE*, 40(9):1098–1101, 1952.

[85] ISO DIS 10918-1 Digital compression and coding of continuous-tone still images (JPEG). CCITT Recommendation T.81.

[86] Van Jacobson. *Session Directory*. Lawrence Berkeley Laboratory. Software on-line[5].

[87] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, Stanford, CA, August 1988.

[88] Van Jacobson. SIGCOMM '94 Tutorial: Multimedia conferencing on the Internet, August 1994.

[3] ftp://cs.ucl.ac.uk/mice/sdr/
[4] http://www.cs.ucl.ac.uk/mice/codec_manual/doc.html
[5] ftp://ftp.ee.lbl.gov/conferencing/sd

[89] Van Jacobson and Steve Deering. Administratively scoped IP multicast. In *Proceedings of the 30th Internet Engineering Task Force*, Toronto, Canada, July 1994.

[90] Van Jacobson and Steven McCanne. *Visual Audio Tool*. Lawrence Berkeley Laboratory. Software on-line[6].

[91] Van Jacobson, Steven McCanne, and Sally Floyd. A privacy architecture for lightweight sessions, September 1994. ARPA Network PI Meeting presentation [7].

[92] Jeffrey M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[93] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall International, Inc., 1989.

[94] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[95] Sugih Jamin. *A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks*. PhD thesis, University of Southern California, August 1996.

[96] Sugih Jamin, Peter B. Danzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995. ACM.

[97] K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith. Adaptive, best-effort delivery of digital audio and video across packet-switched networks. In *Proceedings of the Third International Workshop on Network and OS Support for Digital Audio and Video*, San Diego, CA, November 1992. ACM.

[98] Hemant Kanakia, Partho P. Mishra, and Amy Reibman. An adaptive congestion control scheme for real-time packet video transport. In *Proceedings of SIGCOMM '93*, pages 20–31, San Francisco, CA, September 1993. ACM.

[99] Gunnar Karlsson. Asynchronous transfer of video. *IEEE Communications Magazine*, 34(8), August 1996.

[100] Gunnar Karlsson and Martin Vetterli. Three-deimensional subband coding of video. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1100–1103, New York, NY, April 1988.

[101] Gunnar Karlsson and Martin Vetterli. Packet video and its integration into the network architecture. *IEEE Journal on Selected Areas in Communications*, 7(5):739–751, June 1989.

[102] Leonid Kasperovich. Multiplication free scaled 8x8 DCT algorithm with 530 additions. In *Proc. SPIE*, volume 2419, pages 105–110. ACM, 1995.

[103] Christopher A. Kent and Jeffrey Mogul. Fragmentation considered harmful. In *Proceedings of SIGCOMM '87*. ACM, August 1987.

[104] F. Kishino, K. Manabe, Y.Hayahi, and H. Yasuda. Variable bit-rate coding of video signals for ATM networks. *IEEE Journal on Selected Areas in Communications*, 7(5):801–806, June 1989.

[6] ftp://ftp.ee.lbl.gov/conferencing/vat

[7] ftp://ftp.ee.lbl.gov/talks/lws-privacy.ps.Z

[105] Isidor Kouvelas, Vicky Hardman, and Anna Watson. Lip synchronisation for use over the Internet: Analysis and implementation. In *Proceedings of GLOBECOM '96*, London, UK, November 1996.

[106] Murat Kunt and Ottar Johnsen. Block coding of graphics: A tutorial review. *Proceedings of the IEEE*, 68(7):770–786, July 1980.

[107] Didier Le Gall. Sub-band coding of images with low computational complexity. In *Proceedings of the Picture Coding Symposium*, Stockholm, Sweden, June 1987.

[108] Didier Le Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[109] D. J. LeGall, H. Gaggioni, and C. T. Chen. Transmission of HDTV signals under 140 Mbits/s using a subband decomposition and Discrete Cosine Transform coding. In L. Chiariglione, editor, *Signal Processing of HDTV*, pages 287–293. Elsevier, Amsterdam, 1988.

[110] Krisda Lengwehasatit and Antonio Ortega. Distortion/decoding time trade-offs in software DCT-based image coding. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Munich, Germany, April 1997.

[111] A. S. Lewis and G. Knowles. Image compression using the 2-D wavelet transform. *IEEE Transactions on Image Processing*, 1(2):244–250, April 1992.

[112] Christopher J. Lindblad, David J. Wetherall, and David L. Tennenhouse. The VuSystem: A programming system for visual processing of digital video. In *Proceedings of ACM Multimedia '94*, pages 307–314. ACM, October 1994.

[113] Mark Linton, Parl R. Calder, and John M. Vlissides. InterViews: A C++ graphical interface toolkit. Technical Report CSL-TR-88-358, Stanford University, Palo Alto, CA, July 1988.

[114] Michael R. Macedonia and Donald P. Brutzman. MBone provides audio and video across the Internet. *IEEE Computer*, pages 30–36, April 1994.

[115] Allison Mankin. Random drop congestion control. In *Proceedings of SIGCOMM '90*, Philadelphia, PA, September 1990.

[116] Burt Masnick and Jack Wolf. On linear unequal error protection codes. *IEEE Transactions on Information Theory*, IT-3(4), October 1967.

[117] Steven McCanne. A distributed whiteboard for network conferencing, May 1992. U.C. Berkeley CS268 Computer Networks term project and paper.

[118] Steven McCanne. Joint source/channel coding for multicast packet video. Dissertation Proposal, Qualifying Exam, Electrical Engineering and Computer Science Dept., U.C. Berkeley, Berkeley, CA, September 1994.

[119] Steven McCanne and Sally Floyd. *The LBNL Network Simulator*. Lawrence Berkeley Laboratory. Software on-line[8].

[8] http://www-nrg.ee.lbl.gov/ns/

[120] Steven McCanne and Van Jacobson. *vic:* a flexible framework for packet video. In *Proceedings of ACM Multimedia '95*, pages 511–522, San Francisco, CA, November 1995. ACM.

[121] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *Proceedings of SIGCOMM '96*, pages 117–130, Stanford, CA, August 1996. ACM.

[122] Steven McCanne and Martin Vetterli. Joint source/channel coding for multicast packet video. In *Proceedings of the IEEE International Conference on Image Processing*, pages 25–28, Washington, DC, October 1995.

[123] Steven McCanne, Martin Vetterli, and Van Jacobson. Low-complexity video coding for receiver-driven layered multicast. *Accepted for publication in IEEE Journal on Selected Areas in Communications*, 1997.

[124] K. Metin Uz, Martin Vetterli, and Didier LeGall. Interpolative multiresolution coding of advanced television with compatible subchannels. *IEEE Transactions on CAS for Video Technology, Special Issue on Signal Processing for Advanced Television*, 1(1):86–99, March 1991.

[125] Robert F. Mines, Jerrold A. Friesen, and Christine L. Yang. DAVE: A plug and play model for distributed multimedia application development. In *Proceedings of ACM Multimedia '94*, pages 59–66. ACM, October 1994.

[126] Jeffrey Mogul and Steve Deering. *Path MTU Discovery*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, November 1990. RFC-1191.

[127] F. W. Mounts. A video encoding system with conditional picture-element replenishment. *Bell Systems Technical Journal*, 48(7):2545–2554, September 1969.

[128] Arun Netravali and Barry Haskell. *Digital Pictures*. Plenum Press, New York, NY, 1988.

[129] Antonio Ortega, Kannan Ramchandran, and Martin Vetterli. Optimal trellis-based buffered compression and fast approximations. *IEEE Transactions on Image Processing*, 3(1):16–40, January 1994.

[130] John K. Ousterhout. An X11 toolkit based on the Tcl language. In *Proceedings of the 1991 Winter USENIX Technical Conference*, Dallas, TX, January 1991. USENIX.

[131] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[132] Joseph C. Pasquale, George C. Polyzos, Eric W. Anderson, and Vachspathi P. Kompella. Filter propagation in dissemenation trees: Trading off bandwidth and processing in continuous media networks. In *Proceedings of the Fourth International Workshop on Network and OS Support for Digital Audio and Video*, pages 269–278, Lancaster, U.K., November 1993. ACM.

[133] Joseph C. Pasquale, George C. Polyzos, Eric W. Anderson, and Vachspathi P. Kompella. The multimedia multicast channel. *Internetworking: Research and Experience*, 5(4):151–162, December 1994.

[134] Ketan Patel, Brian C. Smith, and Lawrence A. Rowe. Performance of a software MPEG video decoder. In *Proceedings of ACM Multimedia '93*, pages 75–82. ACM, August 1993.

[135] William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[136] J. Postel. *User Datagram Protocol*. Internet Engineering Task Force, USC/Information Sciences Institute, August 1980. RFC-768.

[137] J. B. Postel. *Transmission Control Protocol*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, August 1989. RFC-793.

[138] Calton Pu, Henry Massalin, and John Ioannidis. The synthesis kernel. *Computing Systems*, 1(1):11–32, 1998.

[139] Kannan Ramchandran, Antonio Ortega, K. Metin Uz, and Martin Vetterli. Multiresolution broadcast for digital HDTV using joint source/channel coding. *IEEE Journal on Selected Areas in Communications*, 11(1):6–23, January 1993.

[140] Anup Rao, Rob Lanphier, et al. Real time streaming protocol (RTSP), October 1996. Internet Draft (work in progress).

[141] K. R. Rao and P. Yip. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic Press, Inc., 1990.

[142] Amy R. Reibman. DCT-based embedded coding for packet video. *Signal Processing: Image Communication 3*, pages 231–237, 1991.

[143] Mark Roseman and Saul Greenberg. GroupKit: A groupware toolkit for building real-time conferencing applications. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, October 1992.

[144] Lawrence Rowe et al. *Continuous Media Toolkit (CMT)*. University of California, Berkeley. Software on-line[9].

[145] Lawrence A. Rowe, Ketan D. Patel, Brian C. Smith, and Kim Liu. MPEG video in software: Representation, transmission, and playback. In *High Speed Network and Multimedia Computing, Symp. on Elec. Imaging Sci. & Tech.*, San Jose, CA, February 1994.

[146] Amir Said and William A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 1996. Submitted for publication.

[147] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), November 1984.

[148] Eve M. Schooler. A multicast user directory service for synchronous rendezvous. Computer science department, California Institute of Technology, September 1996.

[149] Eve M. Schooler and Stephen L. Casner. A packet-switched multimedia conferencing system. *ACM Special Interest Group on Office Information Systems Bulletin*, 10:12–22, January 1989.

[9] http://www.bmrc.berkeley.edu/cmt/

[150] Henning Schulzrinne. Voice communication across the Internet: A network voice terminal. Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[151] Henning Schulzrinne. Dynamic configuration of conferencing applications using pattern-matching multicast. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video*, Durham, NH, April 1995. ACM.

[152] Henning Schulzrinne. *RTP Profile for Audio and Video Conferences with Minimal Control*. Internet Engineering Task Force, Audio-Video Transport Working Group, January 1996. RFC-1890.

[153] Henning Schulzrinne, Steve Casner, Ron Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, Audio-Video Transport Working Group, January 1996. RFC-1889.

[154] Nachum Shacham. Multicast routing of hierachical data. In *Proceedings of the International Conference on Computer Communications*. IEEE, 1992.

[155] Nachum Shacham. Multipoint communication by hierarchically encoded data. In *Proceedings IEEE Infocom '92*, pages 2107–2114, 1992.

[156] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 1948.

[157] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

[158] Robert J. Siracusa, Kuriacose Joseph, Joel Zdepski, and Dipankar Raychaudhuri. Flexible and robust packet transport for digital HDTV. *IEEE Journal on Selected Areas in Communications*, 11(1):88–98, January 1993.

[159] Brian Christopher Smith. *Implementation Techniques for Continuous Media Systems and Applications*. PhD thesis, University of California, Berkeley, December 1994.

[160] Michael F. Speer and Steven McCanne. *RTP usage with Layered Multimedia Streams*. Internet Engineering Task Force, Audio-Video Transport Working Group, March 1996. Internet Draft (work in progress).

[161] Lawrence C. Stewart, Andrew C. Payne, and Thomas M. Levergood. Are DSP chips obsolete? Technical Report CRL 92/10, Digital Equipment Corporation Cambridge Research Lab, Cambridge, MA, November 1992.

[162] David Taubman and Avideh Zakhor. Multi-rate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 3(5):572–588, September 1994.

[163] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26(2):5–18, April 1996.

[164] Thierry Turletti. *INRIA Video Conferencing System (ivs)*. Institut National de Recherche en Informatique et an Automatique. Software on-line[10].

[10] http://www.inria.fr/rodeo/ivs.html

[165] Thierry Turletti and Jean-Chrysostome Bolot. Issues with multicast video distribution in heterogeneous packet networks. In *Proceedings of the Sixth International Workshop on Packet Video*, Portland, OR, September 1994.

[166] Thierry Turletti and Christian Huitema. *RTP Payload Format for H.261 Video Streams*. Internet Engineering Task Force, Audio-Video Transport Working Group, October 1996. RFC-2032.

[167] Thierry Turletti and Christian Huitema. Videoconferencing in the Internet. *IEEE/ACM Transactions on Networking*, 4(3):340–351, June 1996.

[168] Martin Vetterli. Multidimensional subband coding: Some theory and algorithms. *Signal Processing*, 6(2):97–112, February 1984.

[169] Martin Vetterli and Jelena Kovacevic. *Wavelets and Subband Coding*. Prentice-Hall, 1995.

[170] Martin Vetterli and Steven McCanne. On the sub-optimality of receiver-driven layered multicast. Technical report, University of California, Berkeley, CA, January 1997.

[171] Video codec for audiovisual services at p*64kb/s, 1993. ITU-T Recommendation H.261.

[172] Video coding for low bitrate communication, 1996. ITU-T Recommendation H.263.

[173] Mohan Vishwanath and Phil Chou. An efficient algorithm for hierarchical compression of video. In *Proceedings of the IEEE International Conference on Image Processing*, Austin, TX, November 1994.

[174] D. Waitzman, C. Partridge, and S. Deering. *Distance Vector Multicast Routing Protocol*. ARPANET Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, November 1988. RFC-1075.

[175] David Wetherall and Christopher J. Lindblad. Extending Tcl for dynamic object-oriented programming. In *Proceedings of the Tcl/Tk Workshop*, Ontario, Canada, July 1995.

[176] Richard L. White. High-performance compression of astronomical images. In James C. Tilton, editor, *Proceedings of the NASA Space and Earth Science Data Compression Workshop*, Snowbird, Utah, March 1992.

[177] J. W. Woods and S. D. O'Neil. Sub-band coding of images. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(5):1278–1288, May 1986.

[178] Raj Yavatkar and Leelanivas Manoj. Optimistic strategies for large-scale dissemination of multimedia information. In *Proceedings of ACM Multimedia '93*, pages 1–8. ACM, August 1993.

[179] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7:8–18, September 1993.