# Low-complexity Video Coding for Receiver-driven Layered Multicast

Steven McCanne
University of California, Berkeley

Martin Vetterli
EPFL, Switzerland

Van Jacobson
Lawrence Berkeley National Laboratory

January 11, 1997

## Abstract

In recent 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 the development of multipoint audio, video, and shared whiteboard conferencing applications. Because these real-time media are transmitted at a uniform rate to all the receivers in the network, a source must either run at the bottleneck rate or overload portions of its multicast distribution tree. We overcome this limitation 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 different layers across multiple multicast groups and receivers adjust their reception rate by simply joining and leaving multicast groups. In this paper we describe 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 loss resilience (admitting robust operation in loosely controlled environments like the Internet). Even with these constraints, our hybrid DCT/wavelet-based coder exhibits good compression performance. It outperforms all publicly available Internet video codecs while maintaining comparable run-time performance. We have implemented our coder in a "real" application — the UCB/LBL video conferencing tool *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.

## 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* [1]. 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.

Unfortunately, the same problem that plagued the Rolling Stones broadcast constrains other "MBone sessions". To illustrate more clearly the obstacles posed by network heterogeneity, consider the physical network topology that carries live seminars broadcast regularly over the MBone from U.C. Berkeley[1]. Figure 1 depicts this scenario: Some users participate from their office 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. 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

[1] See http://bmrc.berkeley.edu/298/

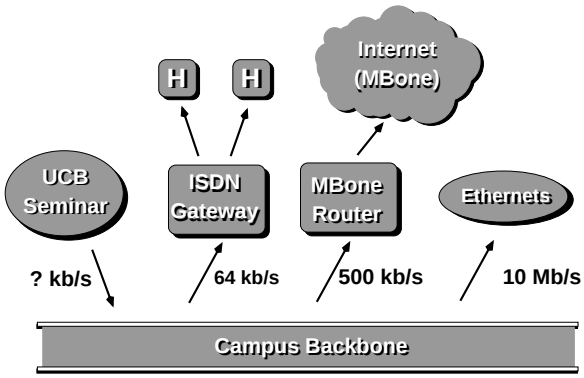


Figure 1: **U.C. Berkeley MBone Seminar.** U.C. Berkeley transmits a multimedia seminar over their campus network, to users at home via ISDN, and over the Internet. A single rate at the source cannot meet the conflicting bandwidth requirements of this heterogeneous set of users.

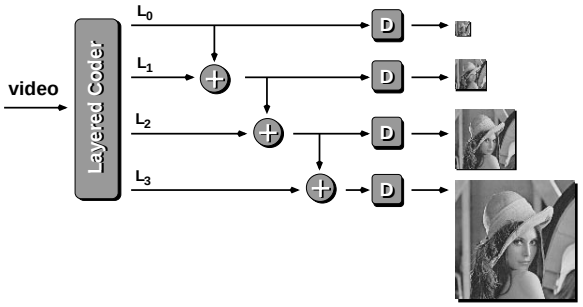


Figure 2: **Layered Video.** A layered codec produces a cumulative set of layers where information is combined across layers to produce progressive refinement. Each decoder module $D$ is capable of decoding any cumulative set of bit strings. Here we show an image at multiple resolutions but the refinement can occur across other dimensions like frame rate or signal-to-noise ratio.

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.

An often cited approach for coping with receiver heterogeneity in real-time multimedia transmissions is the use of layered media streams [2, 3, 4, 5, 6, 7, 8, 9]. 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 adapts 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 subproblems: 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.

## 1.1 Layered Compression

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). This approach, often called *simulcast*, has the advantage that we can use existing codecs and/or compression algorithms as system components. 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 produces the lowest quality version of the signal. If, on the other hand, the decoder receives two layers, it combines 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.

The structure of such a layered video coder is depicted in Figure 2. The input video is compressed to produce a set of logically distinct output strings on channels $L_0$, $L_1$, . . ., and each decoder module $D$ is capable of decoding any cumulative set of bit strings. Each additional string produces an improvement in reconstruction quality.

By combining this approach of layered source coding with a layered transmission system, we can solve the multicast heterogeneity problem [3, 5, 6, 8]. In this architecture, the multicast source produces a layered stream where each layer is transmitted on a different network channel, as illustrated in Figure 3 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, users at home receive only the base layer across their ISDN lines, users in the Internet receive two layers, and users on campus receive all three. Thus each user receives the best quality signal that the network can deliver.

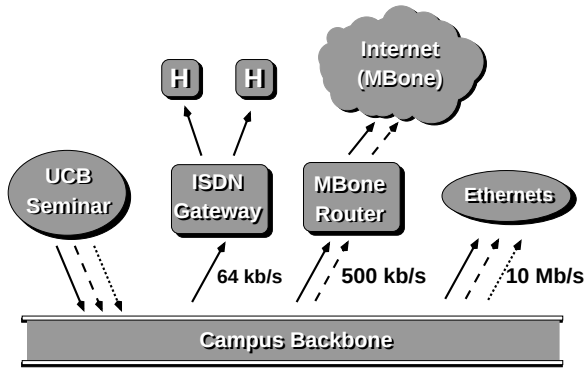


Figure 3: **Layered Transmission.** By combining a layered source coder with a layered transmission system, we solve the heterogeneity problem. The network forwards only the number of layers that each physical link can support.

In this scheme, the network must be able to selectively drop layers at each bottleneck link. While much of the previous work leaves this problem as an implementation detail, a novel and practical scheme was proposed by Deering [2] and was further described and/or independently cited in [10, 3, 4, 5, 8, 11]. In this approach, the layers that comprise the hierarchical signal are striped across distinct IP 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, multicast routers "prune back" that portion of the distribution tree. Although this general mechanism has been discussed in the research community, a system based on this framework had not been deployed because the problem was not studied in detail and specific adaptation protocols that employ the architecture had not been developed. In recent work, we filled this void with a specific protocol we call Receiver-driven Layered Multicast or RLM [12].

A number of research activities have laid the groundwork both for layered video compression [10, 7, 9] and for layered transmission systems [13, 14, 2, 15, 16, 8]. 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 a design and implementation of a comprehensive system for scalable multicast video distribution in heterogeneous networks.

In this paper we give a high level description of our layered transmission system based on RLM to motivate the detailed discussion of our layered coder. In the next section, we sketch the RLM architecture. Subsequently, we describe our layered video compression algorithm based on hybrid DCT/wavelet transform coding and hierarchical conditional replenishment. Next we describe the packetization protocol and receiver recovery strategies. Finally, we report on implementation status and deployment, and conclude.

## 2 Receiver-driven Layered Multicast

In this section, we give a high-level sketch of our Receiver-driven Layered Multicast scheme to establish design constraints on and motivation for a new layered codec. Details of RLM are presented in [12] and [17].

RLM operates within the traditional Internet Protocol architecture and relies upon the delivery efficiency of IP Multicast [18]. It does not require real-time traffic guarantees and assumes only best-effort, multipoint packet delivery. A key feature of IP Multicast is the level of indirection provided by its *host group* abstraction. Host groups provide a *group-oriented* communication framework where senders need not know explicitly about receivers and receivers need not know about senders. Instead, a sender simply transmits packets to a "group address" and receivers tell the network (via the Internet Group Management Protocol or IGMP [19]) that they are interested in receiving packets sent to that group. Moreover, the process by which receivers join and leave these multicast groups is efficient and timely (on the order of a few milliseconds).

Figure 4 illustrates how the group membership protocol can be used to dynamically induce selective forwarding of layers. In this example, 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.

By complementing a layered compression algorithm with the mechanism described above to configure selective forwarding of flows, we move the burden of rate-adaptation from the source to the receivers. 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 multicast groups. Conceptually, each receiver runs the following simple control loop:

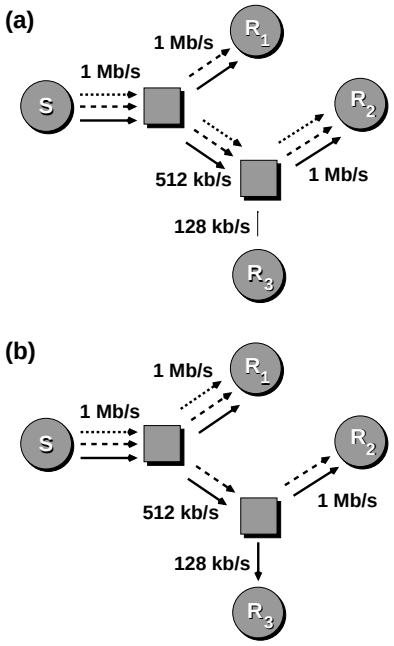


Figure 4: **End-to-end Adaptation.** Receivers join and leave multicast groups at will. The network forwards traffic only along paths that have downstream receivers. In this way, receivers define multicast distribution trees implicitly through their locally advertised interest. A three-layer signal is 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).

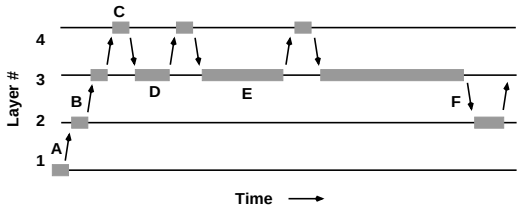


Figure 5: **An RLM "Sample Path".** This diagram illustrates the basic adaptation strategy from the perspective of a given receiver. Initially, the receiver joins the base layer and gradually adds layers until the network becomes congested (C). Here, the receiver drops the problematic layer and scales back its join-experiment rate for that level of subscription.

- 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 [20]. The receiver adds layers until congestion occurs and backs off to an operating point below this bottleneck.

Although a receiver can easily detect that the network is congested by noting gaps in the sequence space of the inbound packet stream, it cannot so easily determine when spare bandwidth becomes available. If only passive monitoring of the inbound packet stream is carried out, differentiating between an inbound rate that is just below capacity and one that is far below capacity is impossible. Instead, RLM uses spontaneous "join experiments" to probe for spare bandwidth. That is, a receiver occasionally tests for residual bandwidth by experimentally adding a layer. If this experiment causes congestion, then the receiver reacts by exponentially scaling back the rate at which it conducts join-experiments for that layer in the future. Over time, a receiver learns that certain levels are problematic while others are not. By running join-experiments infrequently when they are likely to fail, but readily when they are likely to succeed, we minimize their adverse effects.

Figure 5 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 proved 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 congestion occurs (C). As a result, a queue builds up and causes packet loss. When the receiver detects this loss, 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 and the join-timer is further in-

creased (E). Later, unrelated transient congestion provokes the receiver to drop down to layer 2 (F). At this point, because the layer 3 join-timer is still short, the layer is quickly reinstated.

If each receiver runs this adaptation algorithm independently, the protocol would break down at large scales because join-experiments would occur often and cause frequent congestion. Instead, RLM augments its adaptation scheme with "shared learning", where receivers learn from other receivers' failed join-experiments. Details of the shared learning algorithm are described in [12].

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. 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 [21]. 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 [22], minimization of the total distortion in general requires an exchange of information among receivers.

# 3 The Compression Algorithm

Now that we have described the RLM framework, we address the design of a video compression algorithm that complements RLM. To this end, 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 [23], 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 [7]. 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 [9] 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 [24]. 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 [24, 25], 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.

## 3.1 Temporal Compression

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. Several existing Internet video tools use this approach (e.g., our tool *vic* [24], the Xerox PARC Network Video *nv* [26] and Cornell's *CU-SeeMe* [27]) and some commercial H.261 codecs send "block skip codes" for static blocks.

Figure 6 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, H.263, or MPEG employ temporal prediction to achieve higher compression performance. 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 [28, 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 [29, 30]. 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

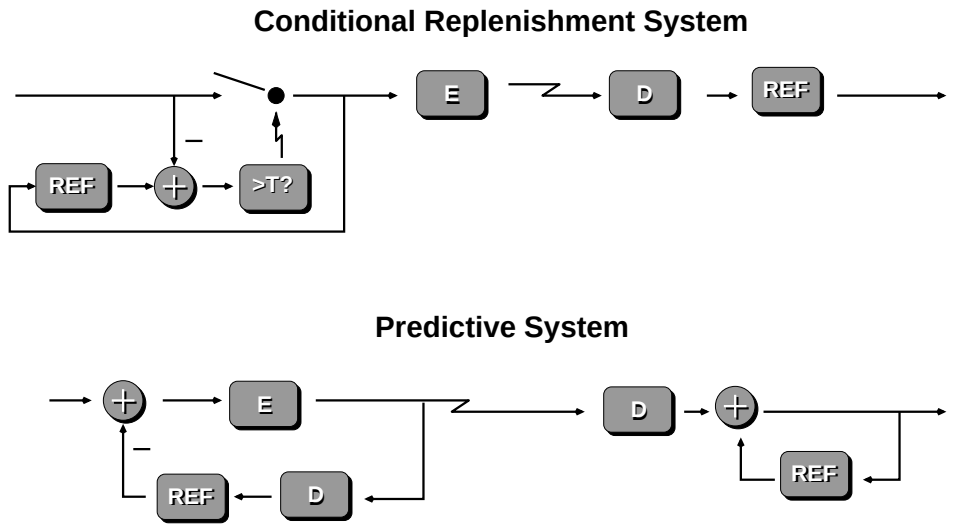


Figure 6: **Temporal Compression Models.** A conditional replenishment system encodes and transmits blocks as independent units, while a predictive system encodes and transmits the residual error between a prediction and the input signal.

the expense of decreased accuracy [31]. Likewise, the degree of quantization applied to the DCT coefficients can be dynamically manipulated to meet a computation budget [32].

- **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.

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. [17] presents evidence that for certain signals and packet loss rates, conditional replenishment outperforms traditional codecs based on temporal prediction.

We now describe the major components of our conditional replenishment algorithm: 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* [26].

### 3.1.1 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).

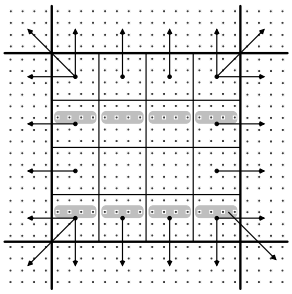

Figure 7: **Block Selection Algorithm.** Block selection is carried out on a 4x4 grid (thin lines) that determines if the containing 16x16 block (thick lines) is replenished. As indicated by the arrows, updates are spread to adjacent 16x16 blocks to minimize "small motion" artifacts.

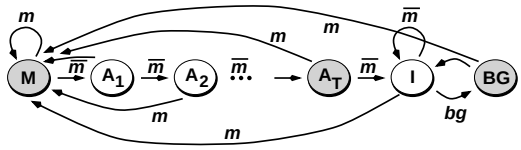


Figure 8: **Block Aging Algorithm.** A separate finite-state machine is maintained for each block in the image. State transitions are based on the presence ($m$) or absence ($\overline{m}$) of motion within the block. A background fill process spontaneously promotes a small number of idle blocks to the background state (*bg*). The block is replenished in the shaded states.

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 7 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.

### 3.1.2 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 8. 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 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.

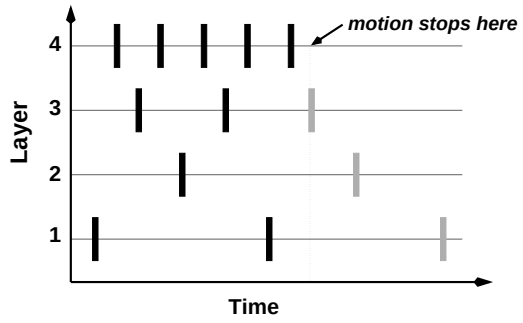


Figure 9: **Temporal Layering.** We extend the conditional replenishment algorithm to produce multiple rates by striping block updates across different output layers. When a block becomes idle, we "slide it" down the layer hierarchy to guarantee that the most up-to-date version appears on the base layer.

### 3.1.3 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 rightmost non-zero bit in the binary representation of $n$.

The hierarchy that results in the case for $M = 4$ is shown in Figure 9. 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 9, 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.

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. [17] contains a detailed analysis of this overhead.

## 3.2 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 [5] 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) [33]. This coding structure is illustrated in Figure 10.

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

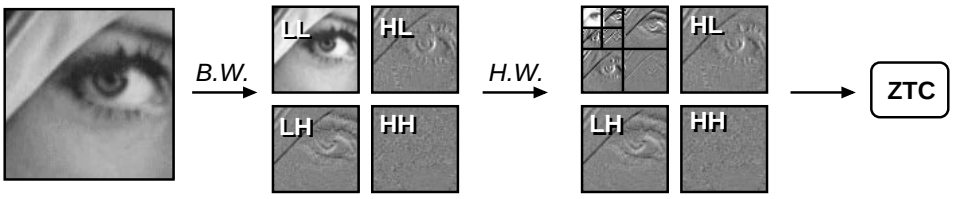


Figure 10: **Zerotree Wavelet Coding Structure.** We decompose a pixel block using our 1/3/3/1 4-tap biorthogonal wavelet (*B.W.*), and in turn, transform the LL subband with a Haar wavelet (*H.W*). The resulting subband coefficient hierarchy is entropy-coded using zerotrees (ZTC).

the image, conditional replenishment assumes spatially confined pixel blocks. 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 [34]:

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

with the following synthesis[2]

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

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 [24]. 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.

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

### 3.2.1 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 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 [35]. 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 11.

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 [36, 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

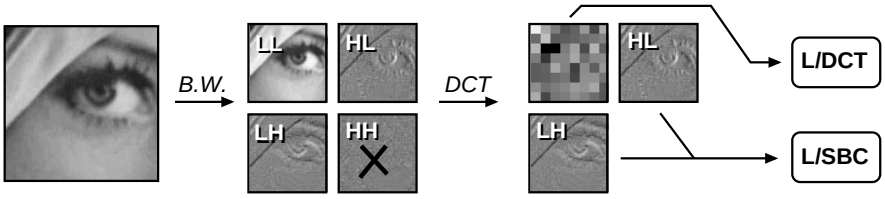


Figure 11: **Hybrid Transform Coding Structure.** We decompose a pixel block using our 1/3/3/1 4-tap biorthogonal wavelet (*B.W.*), and in turn, transform the LL subband with a DCT. The resulting DCT coefficients are run-length/entropy coded and progressively refined (L/DCT = "layered DCT"). The LH/HL subband coefficients are progressively coded by compressing them a bit-plane at a time using a quad-tree decomposition (L/SBC = "layered subband coefficients").

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 [35], 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 [37] though they applied it to bi-level images without any transformation. The *hcompress* algorithm described in [38] 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 [39]. In addition, 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 of [17].

### 3.2.2 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 12 plots a family of operational distortion-rate curves generated by coding the 512x512 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 online optimization that continually updates quantization mix to reflect the changing signal statistics. By including codes to adaptively adjust the quantization mix at the start of each

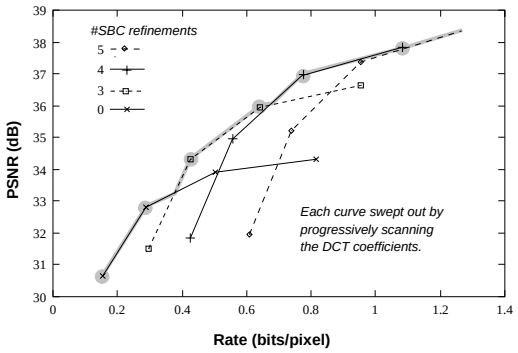


Figure 12: **Bit Allocation.** We determine the best mix of rate across the DCT and subband coefficients by computing the convex hull of a family of curves. Each curve is swept out by progressively scanning the DCT coefficients of the LL subband and each separate curve corresponds to a fixed set of LH/HL coefficient refinement passes.

block, we can perform adaptation on a block granularity. Since the subprocess 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 [40].

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 1. 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[3]. 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

[3] 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 1: Layered Bit Allocation

tested the sensitivity of the optimal choice of quantizers to signal statistics by computing the optimum for several images from the USC 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.

### 3.2.3 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 [33] and progressive-mode JPEG [36, 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 12.

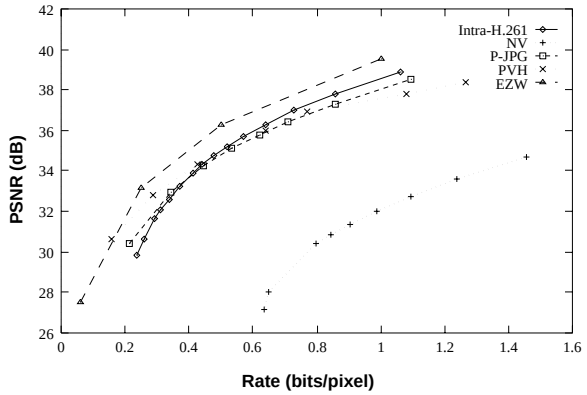


Figure 13: **Relative Compression Performance.** The compression performance of PVH is better than Intra-H.261 at low rates, comparable at medium rates, and somewhat inferior at high rates.

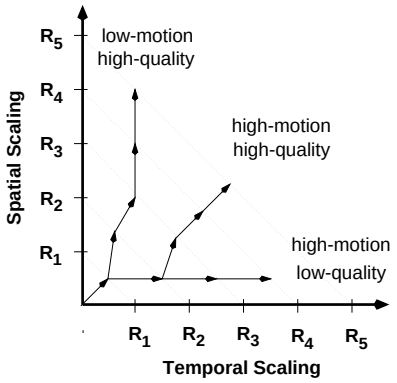


Figure 14: **Temporal/Spatial Scaling.** We cannot scale the spatial and temporal qualities simultaneously. Instead, we must choose a single path through this rate-scaling space. We show three such paths: The lower path leads to a high-motion/low-quality signal, the upper path leads to a low-motion/high-quality signal, and the middle path is a compromise between the two.

- **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 [33].

Figure 13 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.

## 3.3 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 14 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

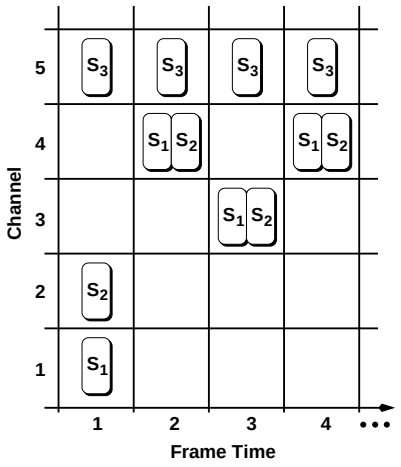


Figure 15: **Spatio-temporal Layering.** We combine layered conditional replenishment with the spatial compression algorithm to induce a spatio-temporal hierarchy where the allocation of spatial layers to network channels evolves over time.

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 $S_1 \ldots S_N$ be a set of spatial layers and $L_1(n) \ldots L_M(n)$ 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 15. 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 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 inter-frame 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.

## 3.4 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 [41] 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 [24]. 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 runtime 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).

# 4 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.

In 1990 Clark and Tennenhouse recognized that this problem could be solved if application semantics were reflected in the design of an application's network protocol. Their Application Level Framing (ALF) protocol architecture [42] 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. About the same time that ALF emerged, we and others developed a number of tools to explore the problem of interactive audio and video transport across packet-switched networks [43, 44, 45, 46, 47, 48]. After several iterations of protocols and experimentation with audio and several different video compression formats, it became clear that a "one size fits all" protocol was inadequate [49, 24]. Instead, a framework based on ALF emerged where a "thin" base protocol defines the core mechanisms and profile extensions define application-specific semantics. 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 [50] and developed a profile for Audio and Video conferences with minimal control [51] along with a number of payload format standards for specific applications like H.261, JPEG, MPEG, etc.

## 4.1 The Real-time Transport Protocol

RTP defines much of the protocol architecture necessary for video transmission over a multipoint packet network. An RTP "session" represents a collection of two or more end systems sending data and control information to each other over two distinct underlying transport channels. For UDP [52] over IP Multicast, these two underlying transport channels are mapped onto two distinct UDP port numbers sharing a common IP multicast group 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. Since Source-ID's can change dynamically (because of conflicts), the "canonical name" or CNAME provides a persistent and globally unique identifier. Data packets are identified only by Source-ID and the RTCP control messages contain 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 reconstruct the sender's original timing while minimizing delay. This "playback point algorithm" can be extended to carry out cross media synchronization [53] by aligning each individual media with the media that has the maximal playback point.

Unfortunately, RTP has no notion of layered streams. In particular, the use of multiple IP multicast addresses in RLM requires that the layered bit stream be striped across distinct RTP sessions. An effort is currently underway — based in part on the work presented in this paper — to modify RTP to allow a single session to span multiple underlying network channels [12, 54]. Our proposed change is an extension to RTP that allows a participant to use one Source-ID consistently across the logically distinct RTP sessions comprising the hierarchy. Accordingly, we run the Source-ID allocation and collision detection algorithm only on the base layer, and likewise, transmit sender identification information only on

the base layer. This proposal is currently under review by the IETF[4] [54].

## 4.2 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*. In this previous work, we developed a codec based on a subset of the H.261 standard, called Intra-H.261, that uses only "intra-coding" of conditionally replenished blocks [24]. 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.

**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 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 1. 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 16 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

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

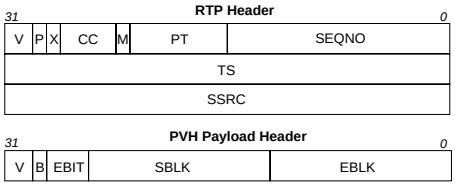


Figure 16: **RTP/PVH Packet Headers.** The RTP header contains a version number (V), a padding bit (P), an extension bit (X), a count of "contributing sources" (CC), i.e., for audio mixing or video compositing, a marker bit (M), a payload type (PT), a 16-bit sequence number (SEQNO), a media-specific 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).

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.

Figure 17 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 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 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.

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.

# 5 Implementation Status

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 [55] multimedia toolkit used to build *vic*. We implemented the RLM protocol in our network simulation testbed [56] and carried out a simulation

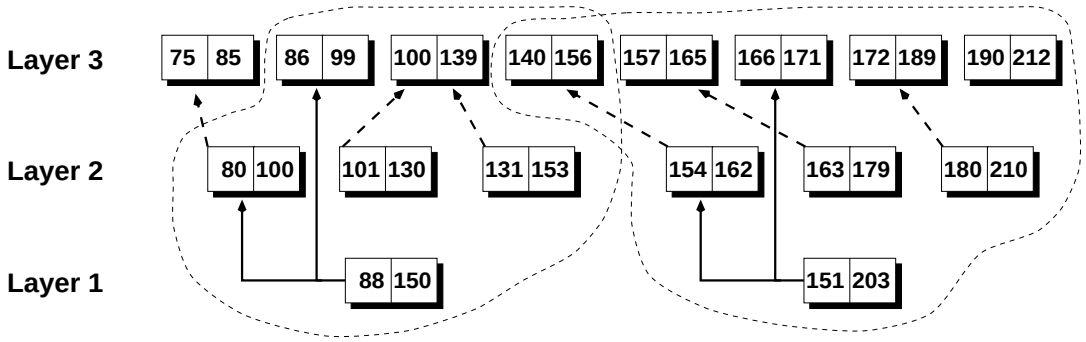


Figure 17: **Sample PVH Packet Stream.** Each base-layer packet defines a range of macroblocks that comprise a slice. Here, we show two slices, each enclosed by a dashed line, that are spread across the layer hierarchy.

study reported in [12, 17].

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.

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) [57, 58], 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 [4].

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 widespread deployment of administrative multicast scope 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.

# 6 Summary

In this paper, we proposed a framework for the transmission of layered signals over heterogeneous networks using a receiver-driven adaptation protocol, RLM. 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 the current practice in 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 the complete systems design and implementation. Together, RLM and PVH provide a comprehensive solution for scalable multicast video transmission in heterogeneous networks.

# 7 Acknowledgments

Elan Amir's work on his "Layered-DCT" algorithm [39] inspired our approach to decomposing the LL subband with a DCT and progressively coding transform coefficients. Elan Amir, Hari Balakrishnan, and Deana McCanne provided thoughtful comments on drafts of this paper. We thank the anonymous reviewers for their excellent feedback. Support for this work was provided by the the Director, Office of Energy Research, Scientific Computing Staff, of the

U.S. Department of Energy under Contract No. DE-AC03-76SF00098 and by the National Science Foundation under grant MIP-93-21302. Equipment grants and additional support were provided by Sun Microsystems, Digital Equipment Corporation, Silicon Graphics Inc., and Phillips.

# Appendix

The rate control of the various multicast groups is done independently of one another, which turns out to be suboptimal in general.

Call $S_i, i = 1...N$ the various multicast sources in the network, $U_{i,j}$ the $j$-th user or receiver in the $i$-th multicast group, and $M_i$ the number of users in multicast group $i$.

Each source has an associated convex distortion rate function $D_i(R)$ which is a priori different for each source. Call $R(U_{i,j})$ the rate received by user $U_{i,j}$. Thus, the $i$-th multicast group produces a total distortion of

$$\hat{D}_i = \sum_{j=1}^{M_i} D_i(R(U_{i,j})) \tag{1}$$

and the overall distortion of all the multicast groups is thus

$$\hat{D} = \sum_{i=1}^{N} \hat{D}_i. \tag{2}$$

In general, the different groups could be weighted differently, which we skip here for simplicity.

On a particular link from node $k$ to $l$ there is bandwidth $B_{k,l}$ available of which a portion $B_{k,l,i}$ is allocated to source $i$, where

$$B_{k,l} \geq \sum_{i=1}^{N} B_{k,l,i}. \tag{3}$$

The goal of an optimal bandwidth allocation is to minimize $\hat{D}$ under the constraint (**??**) on each link in the network.

**Proposition 1** *Independent rate allocation for different multicast groups does not in general minimize total distortion.*

It suffices to construct a counter-example. Take the simplest possible case of two sources $S_1$ and $S_2$ sharing a critical link with bandwidth $B$ to cater to two users $U_{1,1}$ and $U_{2,1}$. Assume the two sessions start simultaneously, so that they will, through competition, each get bandwidth $B/2$ on the critical link. Now, unless

$$\left.\frac{\partial D_1(R)}{\partial R}\right|_{R=B/2} = \left.\frac{\partial D_2(R)}{\partial R}\right|_{R=B/2} \tag{4}$$

(which is the usual optimality condition for rate allocation [34]), the solution is suboptimal.

If the two services are similar, they have the same distortion rate function and equality in (**??**) would hold. However, we can then assume that the services start at different times, and thus the solution with equal bandwidth split will not be reached, and we are again suboptimal. □

The conclusion is that reaching an optimal (equal slope) operating point requires exchange of information between the multicast groups.

**Proposition 2** *Maximizing the total transmitted bit rate does not necessarily minimize the total distortion.*

We take the same example as in Proposition **??**. Any split of the bandwidth $B$ into portions $\alpha B$ and $(1-\alpha)B$, $\alpha \in [0,1]$, maximizes the sum of delivered bandwidth to users 1 and 2. Out of this set, only a single specific $\alpha$ minimizes the distortion, namely the one that achieves

$$\left.\frac{\partial D_1(R)}{\partial R}\right|_{R=\alpha B} = \left.\frac{\partial D_2(R)}{\partial R}\right|_{R=(1-\alpha)B} \tag{5}$$

and thus, in general, maximizing delivered rate does not minimize the distortion. □

# References

[1] Michael R. Macedonia and Donald P. Brutzman, "MBone provides audio and video across the Internet," *IEEE Computer*, pp. 30–36, Apr. 1994.

[2] Steve Deering, "Internet multicast routing: State of the art and open research issues," Oct. 1993, Multimedia Integrated Conferencing for Europe (MICE) Seminar at the Swedish Institute of Computer Science, Stockholm.

[3] 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*. ACM, Aug. 1993, pp. 99–104.

[4] Don Hoffman and Michael Speer, "Hierarchical video distribution over Internet-style networks," in *Proceedings of the IEEE International Conference on Image Processing*, Lausanne, Switzerland, Sept. 1996, pp. 5–8.

[5] Steven McCanne and Martin Vetterli, "Joint source/channel coding for multicast packet video," in *Proceedings of the IEEE International Conference on Image Processing*, Washington, DC, Oct. 1995, pp. 25–28.

[6] Nachum Shacham, "Multicast routing of hierachical data," in *Proceedings of the International Conference on Computer Communications*. IEEE, 1992.

[7] David Taubman and Avideh Zakhor, "Multi-rate 3-D subband coding of video," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 572–588, Sept. 1994.

[8] 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, Sept. 1994.

[9] 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, Nov. 1994.

[10] 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, Oct. 1995.

[11] 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*, San Francisco, CA, Mar. 1996, pp. 553–560.

[12] Steven McCanne, Van Jacobson, and Martin Vetterli, "Receiver-driven layered multicast," in *Proceedings of SIGCOMM '96*, Stanford, CA, Aug. 1996, ACM, pp. 117–130.

[13] Gunnar Karlsson and Martin Vetterli, "Packet video and its integration into the network architecture," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 5, pp. 739–751, June 1989.

[14] Mark W. Garrett and Martin Vetterli, "Joint source/channel coding of statistically multiplexed real-time services on packet networks," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 71–80, Feb. 1993.

[15] 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*, Lancaster, U.K., Nov. 1993, ACM, pp. 269–278.

[16] Nachum Shacham, "Multipoint communication by hierarchically encoded data," in *Proceedings IEEE Infocom '92*, 1992, pp. 2107–2114.

[17] Steven R. McCanne, *Scalable Compression and Transmission of Internet Multicast Video*, Ph.D. thesis, University of California, Berkeley, Dec. 1996.

[18] Stephen E. Deering, *Multicast Routing in a Datagram Internetwork*, Ph.D. thesis, Stanford University, Dec. 1991.

[19] W. Fenner, *Internet Group Management Protocol, Version 2*, Internet Engineering Task Force, Inter-Domain Multicast Routing Working Group, Feb. 1996, Internet Draft (work in progress).

[20] Van Jacobson, "Congestion avoidance and control," in *Proceedings of SIGCOMM '88*, Stanford, CA, Aug. 1988.

[21] Jeffrey M. Jaffe, "Bottleneck flow control," *IEEE Transactions on Communications*, vol. 29, no. 7, pp. 954–962, July 1981.

[22] Martin Vetterli and Steven McCanne, "On the suboptimality of receiver-driven layered multicast," Tech. Rep., University of California, Berkeley, CA, Jan. 1997.

[23] Jean-Chrysostome Bolot, "End-to-end packet delay and loss behavior in the Internet," in *Proceedings of SIGCOMM '93*, San Francisco, CA, Sept. 1993, ACM, pp. 289–298.

[24] Steven McCanne and Van Jacobson, "*vic:* a flexible framework for packet video," in *Proceedings of ACM Multimedia '95*, San Francisco, CA, Nov. 1995, ACM, pp. 511–522.

[25] F. W. Mounts, "A video encoding system with conditional picture-element replenishment," *Bell Systems Technical Journal*, vol. 48, no. 7, pp. 2545–2554, Sept. 1969.

[26] Ron Frederick, "Experiences with real-time software video compression," in *Proceedings of the Sixth International Workshop on Packet Video*, Portland, OR, Sept. 1994.

[27] Tim Dorcey, "CU-SeeMe desktop videoconferencing software," *ConneXions*, vol. 9, no. 3, Mar. 1995.

[28] Arun Netravali and Barry Haskell, *Digital Pictures*, Plenum Press, New York, NY, 1988.

[29] Charles Compton and David Tennenhouse, "Collaborative load shedding for media-based applications," *International Conference on Multimedia Computing and Systems*, May 1994.

[30] 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*, Durham, NH, Apr. 1995, pp. 179–182.

[31] Leonid Kasperovich, "Multiplication free scaled 8x8 DCT algorithm with 530 additions," in *Proc. SPIE*. ACM, 1995, vol. 2419, pp. 105–110.

[32] 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, Apr. 1997.

[33] Jerome M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

[34] Martin Vetterli and Jelena Kovacevic, *Wavelets and Subband Coding*, Prentice-Hall, 1995.

[35] 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 *Signal Processing of HDTV*, L. Chiariglione, Ed., pp. 287–293. Elsevier, Amsterdam, 1988.

[36] "ISO DIS 10918-1 Digital compression and coding of continuous-tone still images (JPEG)," CCITT Recommendation T.81.

[37] Murat Kunt and Ottar Johnsen, "Block coding of graphics: A tutorial review," *Proceedings of the IEEE*, vol. 68, no. 7, pp. 770–786, July 1980.

[38] Richard L. White, "High-performance compression of astronomical images," in *Proceedings of the NASA Space and Earth Science Data Compression Workshop*, James C. Tilton, Ed., Snowbird, Utah, Mar. 1992.

[39] Elan Amir, Steven McCanne, and Martin Vetterli, "A layered DCT coder for Internet video," in *Proceedings of the IEEE International Conference on Image Processing*, Lausanne, Switzerland, Sept. 1996, pp. 13–16.

[40] Antonio Ortega, Kannan Ramchandran, and Martin Vetterli, "Optimal trellis-based buffered compression and fast approximations," *IEEE Transactions on Image Processing*, vol. 3, no. 1, pp. 16–40, Jan. 1994.

[41] Ketan Patel, Brian C. Smith, and Lawrence A. Rowe, "Performance of a software MPEG video decoder," in *Proceedings of ACM Multimedia '93*. ACM, Aug. 1993, pp. 75–82.

[42] David D. Clark and David L. Tennenhouse, "Architectural considerations for a new generation of protocols," in *Proceedings of SIGCOMM '90*, Philadelphia, PA, Sept. 1990, ACM.

[43] Van Jacobson and Steven McCanne, *Visual Audio Tool*, Lawrence Berkeley Laboratory, ftp://ftp.ee.lbl.gov/conferencing/vat.

[44] Eve M. Schooler and Stephen L. Casner, "A packet-switched multimedia conferencing system," *ACM Special Interest Group on Office Information Systems Bulletin*, vol. 10, pp. 12–22, Jan. 1989.

[45] Ron Frederick, *Network Video (nv)*, Xerox Palo Alto Research Center, ftp://ftp.parc.xerox.com/net-research.

[46] Steven McCanne, "A distributed whiteboard for network conferencing," May 1992, U.C. Berkeley CS268 Computer Networks term project and paper.

[47] Thierry Turletti, *INRIA Video Conferencing System (ivs)*, Institut National de Recherche en Informatique et an Automatique, http://www.inria.fr/rodeo/ivs.html.

[48] 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.

[49] 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*, Boston, MA, Sept. 1995, ACM, pp. 342–356.

[50] 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, Jan. 1996, RFC-1889.

[51] Henning Schulzrinne, *RTP Profile for Audio and Video Conferences with Minimal Control*, Internet Engineering Task Force, Audio-Video Transport Working Group, Jan. 1996, RFC-1890.

[52] J. Postel, *User Datagram Protocol*, Internet Engineering Task Force, USC/Information Sciences Institute, Aug. 1980, RFC-768.

[53] Van Jacobson, "SIGCOMM '94 Tutorial: Multimedia conferencing on the Internet," Aug. 1994.

[54] Michael F. Speer and Steven McCanne, *RTP usage with Layered Multimedia Streams*, Internet Engineering Task Force, Audio-Video Transport Working Group, Mar. 1996, Internet Draft (work in progress).

[55] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[56] Steven McCanne and Sally Floyd, *The LBNL Network Simulator*, Lawrence Berkeley Laboratory, http://www-nrg.ee.lbl.gov/ns/.

[57] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala, "RSVP: A new resource reservation protocol," *IEEE Network*, vol. 7, pp. 8–18, Sept. 1993.

[58] R. Braden, L. Zhang, D. Estrin, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP) – version 1 function specification," Nov. 1996, Internet Draft (RFC pending).