A REAL-TIME SCALABLE SOFWARE VIDEO CODEC FOR COLLABORATIVE APPLICATIONS OVER PACKET NETWORKS

J. Hartung, A. Jacquin, J. Pawlyk, K.L. Shipley

Multimedia Communications Research Laboratory
Lucent Technologies, Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{hrt,arnaud,jsp,ship}@bell-labs.com

ABSTRACT

Layered coding architectures are attractive in theory for two reasons. First, they naturally allow for heterogeneity in networks and receivers in terms of client processing capability and network bandwidth. Second, they correspond to optimal utilization of available bandwidth when several video quality levels are desired. In practice, the increased complexity imposed by the layering constraint can force the designer to do away with key parts of a coding algorithm, e.g. motion estimation for video coding. In this paper, we propose a scalable sofware-only video codec architecture with motion estimation, which is suitable for real-time audio and video communication over packet networks.

The coding algorithm is compatible with ITU-T recommendation H.263+ and includes various techniques to reduce complexity. Fast motion estimation (ME) is performed at the H.263-compatible base layer and used at higher layers, and perceptual macroblock skipping is performed at all layers before ME. SNR scalability is achieved through rate control performed independently at each layer. Spatial scalability is achieved by interpolation at the decoder. A simple technique is used to detect and conceal lost packets. Error propagation from packet loss is avoided by periodically rebuilding a valid predictor in Intra mode at each layer.

The platform is implemented in two modules: a portable software library and a Windows module which performs Video for Windows capture and display and network interface. The platform supports both unicast and multicast IP sockets. The various video layers are transmitted separately on different IP sockets. Synchronization of the layers is achieved using time stamps in the RTP headers. Up to three video layers can be encoded/decoded in real-time on a 200 Mhz Pentium PC, for input images in QCIF format with a target frame rate up to 7.5 fps.

ACM Multimedia'98, Bristol, UK

1. INTRODUCTION

Although still not widely used today, videoconferencing over packet networks such as the Internet is bound to become widespread in the near future, as PCs come equipped with video capture boards and inexpensive cameras and native codec implementations become widely available. As described in [1], besides the "venerable applications" of videoconferencing such as remote meetings, distance learning and telecommuting, a number of innovative applications of the technology are starting to appear in the fields of tourism, medicine, banking, surveillance and appliance re-

pair. Videoconferencing over packet networks is also opening new markets such as for media call servers [2].

Packet based networks, however, are inherently heterogenous, which offers new challenges in video coding technology. User connections to the public Internet or corporate intranets range from fast Ethernet access to ISDN lines and modems. Network clients also typically have diverse processing capabilities as well as video quality requirements. As explained by McCanne et al. in [3], *scalable* or *layered* video coding algorithms provide a graceful way to accomodate this heterogeneity of IP networks and receivers, when these algorithms are used together with an intelligent subscription algorithm such as RLM [4] which allows for dynamic subscription to video layers based on bandwidth availability/network congestion.

In [3], McCanne et al. suggest that motion-compensated prediction should not be used in layered video coding algorithms because of issues of complexity, error resilience, the desirability to decouple encoder and decoder states, and of compute-scalable decoding. This paper describes a native implementation of a layered algorithm which includes motion-compensated prediction combined with conditional replenishment. Our algorithm preserves the advantages offered by conditional replenishment while preserving the performance offered by motion-compensated prediction. It includes a simple but efficient error recovery scheme which detects and conceals the effects of lost packets. Also, the motion compensation algorithm is low-complexity and is only used at the base layer, which keeps overall algorithm complexity in check. The main features and the performance of the algorithm are described in Section 2. The architecture of our sofware platform is described in Section 3.

2. LAYERED CODEC ALGORITHM

2.1. Codec overview

The encoder produces an embedded video stream consisting of a base layer stream at the lowest rate and up to two enhancement streams at higher rates. This embedded architecture is illustrated in Figure 1, where E_0 denotes the base layer encoder, and E_1, E_2 denote the enhancement layer encoders. Together, E_0, E_1, E_2 make up the embedded 'meta-encoder' E. Similarly, base and side layer decoder modules D_0, D_1, D_2 make up the embedded 'meta-decoder' D. The base layer stream is H.263-compatible [5] and therefore can be decoded by any H.263-compliant decoder. Unlike the codec proposed by McCanne in [3], our base layer encoder uses motion estimation to generate motion compensated predictions. The enhancement layer streams are compatible with ITU-T recommendation H.263+ [6]. The enhancement layer encoders use a mix of predictors: temporally synchronous reconstructed frames at the previous layer (EI mode) as well as motion-compensated predictions

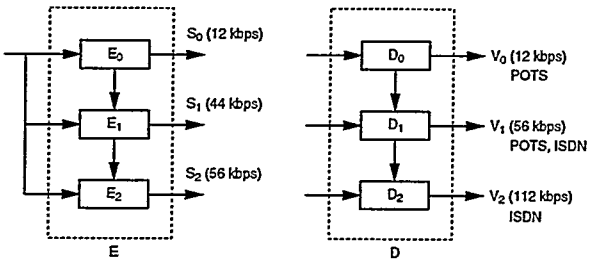

Figure 1: **Block diagram of embedded codec. The encoder generates one base layer stream S_0 and two enhancement layer streams S_1 and S_2.**

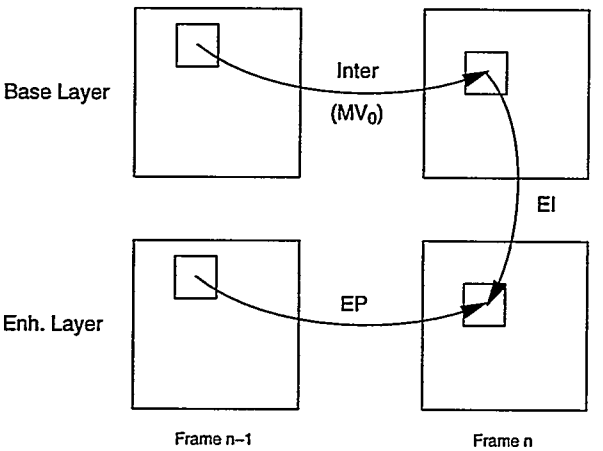

Figure 2: **Prediction and coding modes at base and enhancement layers. MV_0 denotes the motion vector field computed at the base layer**

generated using the motion vectors computed at the base layer (EP mode), illustrated in Figure 2. The coding mode decision is based on estimates of prediction error energy, according to:

```
if (dfd_mad < fd_mad)
  coding_mode = INTER;
else
  coding_mode = EI;
```

where `dfd_mad` is the Mean Absolute Difference (MAD) of the displaced frame difference (with motion vectors obtained at the base layer) and `fd_mad` is the MAD of the difference between original frame and reconstructed frame at the base layer.

2.1.1. Rate control

Rate control is performed independently at each layer. A single target frame rate F is currently specified at the encoder side and used in all layers. In each layer, the capacity

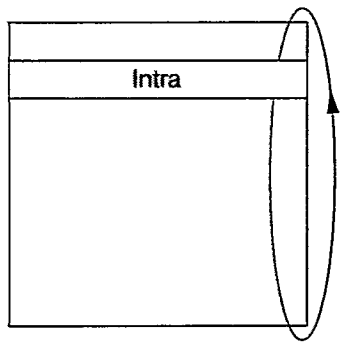

Figure 3: **Periodic encoding of lines of Intra macroblocks for error propagation suppression.**

of a virtual encoder buffer is set to one half of the target bit rate (in bits-per-second). Whenever the buffer reaches 80% of capacity, the upcoming video frame is not encoded (i.e. skipped) in this layer. This simple scheme enables the buffers to remain fairly close to half-full. Our algorithm for selecting the quantization parameter once a video frame is to be coded is directly derived from the MPEG4 Verification Model [7] which models coding rate as a second order polynomial of the form:

$$R = p_1 \cdot S \cdot Q^{-1} + p_2 \cdot S \cdot Q^{-2},$$

where R denotes the coding rate for the frame, S denotes the displaced frame difference energy, Q denotes the quantization parameter and p_1, p_2 are the model parameters.

2.1.2. *Error control*

As will be seen in Section 3, the software platform uses the UDP protocol for packet transmission [8]. Since the UDP protocol does not guarantee packet delivery, the decoder needs to be able to detect packet loss and recover from it. Our algorithm for recovery currently consists of two distinct but related modules. First, the decoder detects lost packets from the sequence field of the RTP packet header. In image regions corresponding to lost macroblocks, the data from the previously reconstructed frame is simply repeated. Second, the encoder forces periodic encoding of lines of macroblock as Intra blocks so as to suppress error propagation which arises from generating motion-compensated predictions using erroneous data. This scheme is illustrated in Figure 3. Assuming a coding rate of F fps, this simple scheme achieves recovery in `mb_lines`$/F$ seconds, where `mb_lines` indicates the number of lines of macroblocks per frame[1].

[1]In a typical situation where $F = 7.5 fps$ and input video images are in QCIF format (176×144), recovery is achieved in 1.2 sec.

2.1.3. *Quality adaptive postfiltering*

The decoder is followed by a postfiltering module which optimally removes quantization artifacts by adapting to a local estimate of decoded image quality. This estimate is provided by the quantization parameter QP and is available on a macroblock basis. Details of the algorithm can be found in [9].

2.2. Complexity reduction

Since our codec is a sofware-only implementation meant to run on PCs, it is important to try and minimize the complexity of the algorithms. For temporal compression, the encoder makes use of two techniques which significantly reduce complexity while preserving video quality: fast motion estimation and perceptual conditional replenishment.

2.2.1. *Fast motion estimation*

As mentioned in section 2.1, motion estimation is performed only at the base layer. H.263+ syntax allows the use of motion vectors at enhancement layers for prediction and our coder uses the base layer motion vectors across layers. The motion estimation algorithm is a block matching algorithm which performs a logarithmic cross-search in the range ± 15 on 16×16 luminance blocks [12, 13].

2.2.2. *Perceptual conditional replenishment*

Conditional replenishment (CR) refers to the ability of a codec to detect and signal image areas which change very little from frame to frame [10]. Standard recommendation H.263 allows for conditional replenishment of macroblocks through block skipping. In our implementation, perceptual CR is performed prior to motion estimation in order to reduce the complexity of that module. We use a measure C as an indicator of change for 8×8 luminance blocks. Let m denote the average luminance of the 8×8 input image block `curr`$[i, j]$. C is computed as follows:

```
for (i=0; i<8; i++) {
  for (j=0; j<8; j++) {
    diff = ABS(curr[i,j] - prev[i,j]);
      if (diff > T[m]) {
        C += diff/T[m];
      }
  }
}
if (C > max_above_T)  change = TRUE;
else  change = FALSE;
```

The parameter `max_above_T` was chosen equal to 6. A 16×16 macroblock is skipped if no 8×8 block indicates

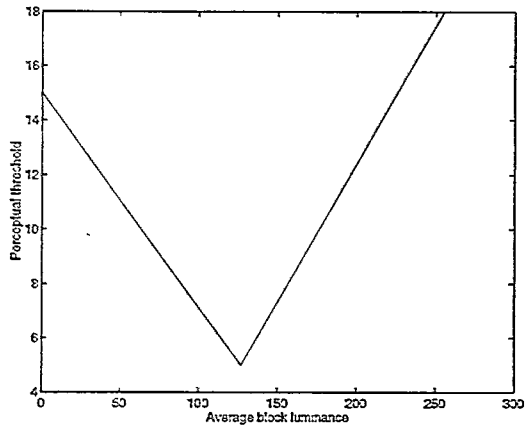

Figure 4: **Perceptual threshols for conditional replenishment.**

significant change. The dependence of the threshold $T[m]$ on the average block luminance m is illustrated in Figure 4. The shape of the graph is based on the work of Chiu and Berger [11] and models the sensitivity of the human eye to spatio-temporal intensity changes as a function of surrounding background intensity.

2.3. Algorithm performance

We compared the relative performances of a single layer platform and a multi layer platform with the following experiments. The single layer platform ran on input sequences in QCIF format (176×120 for luma) stored on disk, at a target frame rate of 7.5 fps and target channel rates of 12, 48, 96, and 144 kbps. These four operating points are converted to bits-per-pixel in Table 1.

The three-layer platform ran on the same sequences, at the same target rate, but with two different rate combinations. The first one specified a base layer rate of 12 kbps and side layer rates of 36 and 48 kbps, which corresponds to cumulative rates of 12, 48 and 96 kbps. The second one specified a base rate of 48 kbps and side layer rates of 48 kbps, corresponding to the cumulative rates of 48, 96, and 144 kbps. Peak signal-to-noise ratios averaged over the three sequences are plotted in Figure 5. The interpolation functions were obtained by fitting to the experimental data a model of the form:

$$\text{PSNR} = a\text{Ln}(R - b) + c$$

where R denotes rate and a, b, c are the model parameters. Three video-conferencing type sequences were used in the experiment: 'Manya' (15s), 'Mother-Daughter' (8s), and 'Sam-Dad' (15s). We can see from Figure 5 that layering in the context of standard-based H.263+ compatible coding comes at a price in terms of image quality. While the single

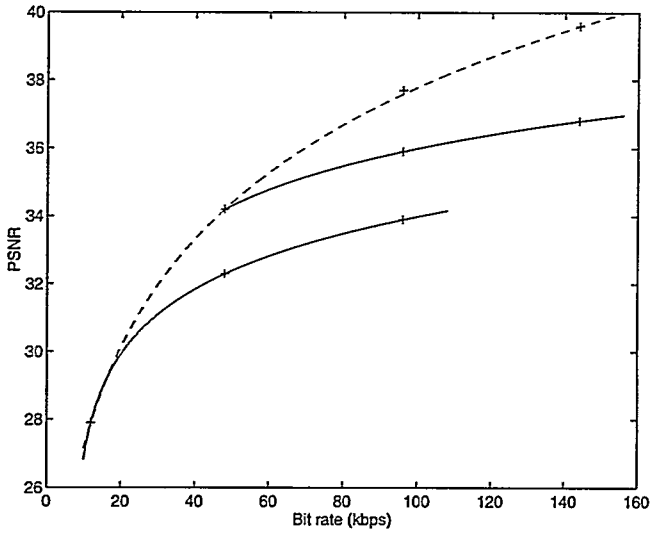

Figure 5: **Rate distortion curves for single layer platform (dashed) and multi layer platform (solid curves).**

channel rates (bps)	12	48	96	144
bits-per-pixel	0.05	0.20	0.40	0.61

Table 1: **Correspondance between channel rates in bps and bits-per-pixel for QCIF sequences in YUV format coded at a target frame rate of 7.5 fps.**

layer and multiple layer graphs start at the same operating point (base layer encoding), they start to diverge significantly from there. These results seem to indicate that the paradigm of multiple encodings of image residuals obtained either from motion-compensation (EP mode) or reconstructions at previous layers (EI mode) may be far from optimal. It would be of great interest to compare these results to those obtained under significantly different coding paradigms.

3. PLATFORM OVERVIEW

3.1. Software architecture

The sofware platform consists of an encoder and a decoder, each implemented as a multithreaded Windows™ application. Each process is organized into two threads; a thread with Windows™ API calls, video capture or display, and a second thread that implements calls to the codec. This organization allows us to use the Windows™ thread-scheduling mechanism to manage scheduling of capture, coding and user input to the application controls. It also keeps parts of the code that are callable under different environments (Windows™, Unix™) separated from OS and hardware specific elements of the platform.

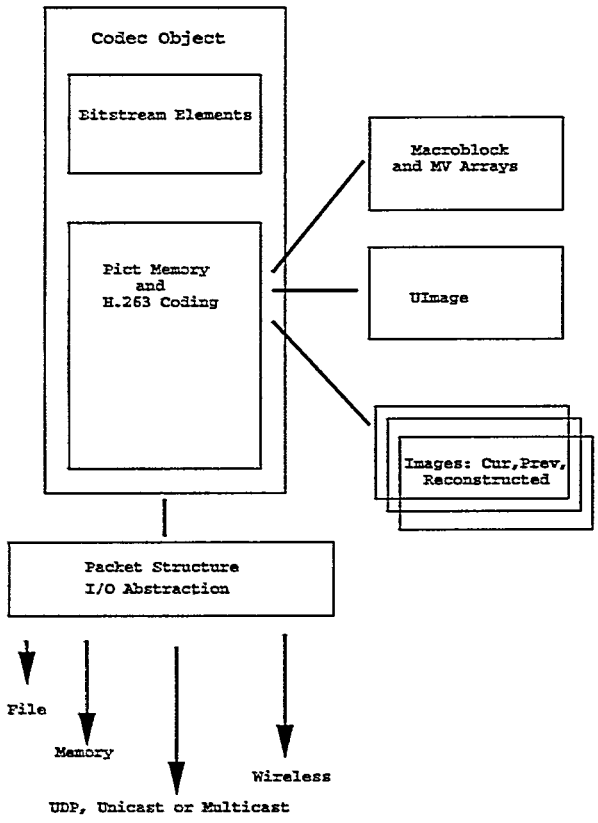

Figure 6: **Pict Object organization.**

3.1.1. Object oriented platform design

The platform was written in the C Programming Language for portability, but we used an Object Oriented approach in its design and implementation. Each significant element of the code was built around a C structure with well-defined routines to create, destroy and configure the structure memory, and all structure elements were accessed through Macros. Each of these 'objects' was tested for memory leaks and basic functionality before being incorporated into the platform. After an object was built and tested, we added functions to implement codec routines, naming conventions for functions (i.e.: ObjectFunctionImplementation ()), and organized each object's primary code into a separate file to help maintain object boundaries in the platform. The design allowed new memory elements and new functionality to be added to an object and allowed new objects to be added to the platform with little or no reengineering.

Figure 6 shows the organization of the Pict object which consists of a core that is generally organized like Recommendation H.263, subobjects for image and data storage, and a Packet object that handles all of the bitstream I/O functions. The top level object, labeled codec in the figure, was designed to be interchanged with other kinds of codecs. For example, a simple MPEG4 VOP was built and able to use most of the support code designed for the H.263 codec. Other designs such as wavelet based codecs have also been considered.

Figure 7 shows the packet structure. This object abstracts all of the I/O functions under a simple and consistent interface. Modules that are chosen at compile time allow the same code to output data to file, IP network, wireless network or memory buffer. The Packet code also supported a header object that allowed easy management and insertion of extra information, including RTP headers.

With the highly organized and modular design of the H.263 codec, it was relatively straightforward to create multiple instances of the Pict object, write a few special functions to support multiple layers and create an H.263+ codec as shown in Figure 8.

3.1.2. IP network interface

Although several I/O interfaces were implemented, this platform was primarily intended to provide communication between an encoder and multiple decoders over a packet based network. The Windows Sockets Application Programming Interface (WinSock API) [14] was used for the IP network interface because it is widely available, supported by Windows and designed to be invoked via C applications. The UDP protocol was specified for the socket connections because it provides the high data transmission rates needed for real-time performance. However, UDP does not guarantee packet delivery. As previously discussed in Section 2.1.2, the codec algorithm detects packet loss and recovers from it. Both the unicast and multicast models for packet transmission were implemented. Multicasting allows a packet sent by the encoder to reach multiple decoders. However, not all networks support multicasting, so unicasting, where an encoder transmits each packet to each decoder individually, was also implemented.

The platform specifies a socket for each video layer. Each decoder sends a 'connect request' for each desired video layer on the appropriate socket. The encoder maintains the network information for each decoder in a separate linked list for each requested layer. 'Disconnect requests' from the decoders received on these sockets cause the encoder to remove the entries from the appropriate linked lists. This allows the decoders to request as many enhanced layers as they can handle and to scale up or down as network conditions change.

When a packet is ready for transmission, the encoder checks the entries in the linked list for the appropriate encoding layer. The network information in each entry is used to transmit the packet over a socket to each decoder. In the multicast case, the packet is only transmitted once, to the previously agreed-upon multicast network address.

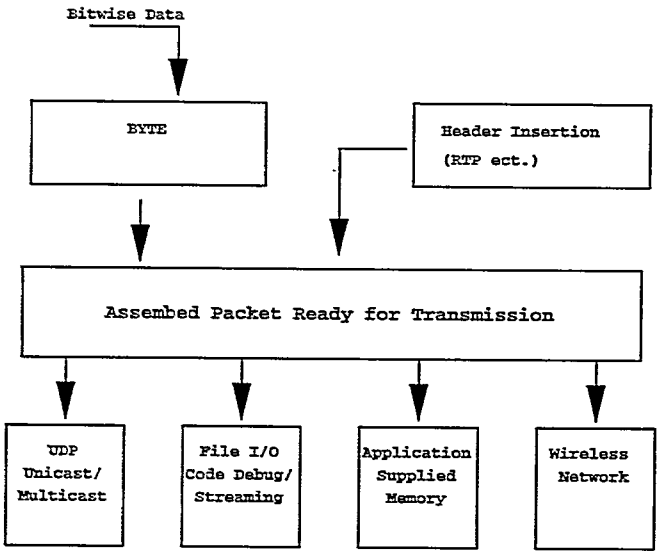

Figure 7: **Packet structure.**

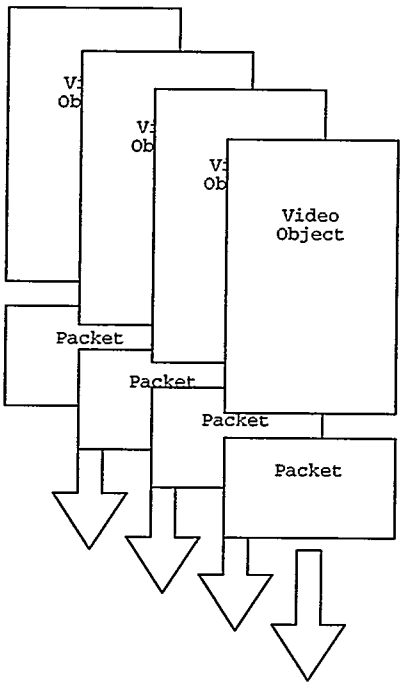

Figure 8: **Sofware organization of H.263+ codec.**

The decoders monitor their socket connections for incoming packets, to be processed promptly.

3.2. Synchronization of video layers

One of our goals for the platform was to minimize delay so it could be used for collaborative applications over packet-based networks. We therefore chose not to implement a long buffer for packet reorganization. At each video layer the decoder collects packets until packets from the next time frame are received, until all of the information needed to reconstruct a layer is received, or until some other condition that forces reconstruction and display of the frame in the current time slice is met. As each packet in the current time slice is received, it is parsed into the Pict structure corresponding to its video layer. This design only reorders packets within the current time slice, but reordering within the current time slice seems to be a reasonable assumption on high quality networks such as our local corporate LAN. We also have some evidence that although out of order packets are common on the Internet, they are most commonly out of order by small amounts [15].

There is a tradeoff between sending the largest packets the network can tolerate without fragmentation and sending smaller packets with the extra overhead in IP, RTP and other headers that are sent with each packet of compressed video data. At bit rates below 128 kbps and frame rates around 7.5 fps, most video frames fit in one or two packets. Some preliminary work has shown that forcing frames into extra packets gives better quality in the presence of errors, but more experiments are needed to really understand the trade-offs between packet size, bit rate and network conditions.

3.3. Packet synchronizer algorithm

Because of the embedded structure of the decoder, video frames at various quality levels are reconstructed from layer frames corresponding to the same temporal reference. Since rate control is performed independently at each layer, some of these layers may be missing as is illustrated in Figure 9. Implementing a codec without a module to reorder packets was simple for the single layer case. The multiple layer case was more complicated. Our design divided the packet processing stage into three independent parts as illustrated by the pseudocode algorithm in Figure 10.

The first stage (ReadPorts) reads packets from a socket if the port for the layer is not stalled, and stores the RTP timestamp of the received packet. The second stage (ProcessPorts) checks the timestamp of the received packet. If the timestamp matches the next display time, the packet is parsed into the Pict Structure. If the timestamp is later than the current display time, the layer is put into the stalled state, which stops subsequent reads. If the timestamp is before the current display time, the packet is dropped. Each layer maintains its own 'stalled' state and time. Each layer also maintains a 'complete' state variable that signals the Reconstruct stage that the Port Processing stage has determined that the layer is complete, or will not be completed.

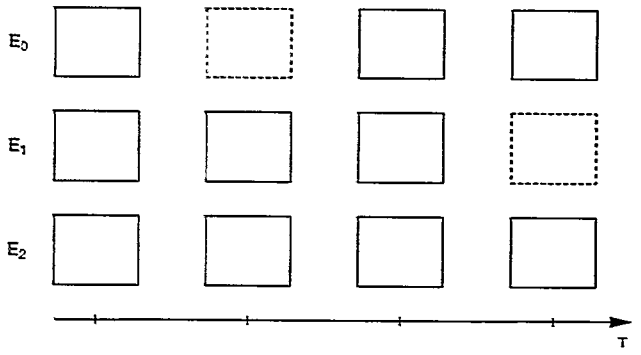

Figure 9: **Base and enhancement frames generated by layered encoder. Missing layers are indicated by dotted rectangles.**

The third stage (Reconstruct) determines whether the current collected data should be decoded and displayed. After the frame is displayed, all ports are taken out of the complete state, and the next display time is set from the earliest next timestamp received from any socket. The pseudocode example does not detail startup, detection of rollover in the RTP clock, nor the steps involved in the Reconstruction decision.

4. CONCLUSION

This work described in detail a native PC implementation of a scalable sofware video codec platform. The platform is modular, supports several transmission modes (file I/O, IP unicast and multicast, wireless network), and is rate-controlled to allow constant bit rate encoding.

The coding algorithm is compatible with recommendation H.263+ and supports up to three video layers. It includes motion estimation at the base layer, a number of complexity reduction schemes, as well as a simple error recovery scheme for unreliable packet-based transmission. Coding simulations on stored input sequences show that the layering constraint results in loss of performance as illustrated by rate-distortion curves obtained from running single and multi layer platforms at various rates.

5. REFERENCES

[1] M. Pihlman, Transforming DVC from a Tool into a Solution. *Desktop Video Communications*, Sept-Oct 1997.

[2] A. Sulkin, Media Call Servers: New Product for an Emerging Market. *Desktop Video Communications*, Sept-Oct 1997.

[3] S. McCanne, M. Vetterli, V. Jacobson, Low-complexity Video Coding for Receiver-driven Layered Multicast. *Trans. JSAC*, vol. 16, no. 6, pp. 983-1001, August 1997.

```
ReadPorts{

    For (All Layers  X){
        If (LayerXStalled == FALSE){
            Read Socket X;
            GetHeader From Packet;
            NextDisplayTimeNext = HeaderTimestamp;
         }
    }
}

ProcessPorts{

    For (All Layers  X){
        if(LayerXHeaderTime == NextDisplayTime){
            ParsePacket_To_Pict_Layer1Structure;
            LayerXStalled = FALSE;
            if (PictLayerXComplete)
                LayerXComplete = TRUE;
        }

        else if (LayerXHeaderTime > NextDisplayTime)
            LayerXStalled = TRUE;
            LayerXComplete = TRUE;
        }

        else{
            LayerXStalled = FALSE;
        }
    }
}

Reconstruct{

    if (AllLayersComplete || ForceReconstruction){
        ReconstructCurrentFrame;
        DisplayCurrentFrame;
        UpdatePlatformTime;
            for (All Layers X)
                LayerXComplete = FALSE;
    }
}
```

Figure 10: **Pseudo Code For Packet Synchronization.**

[4] S. McCanne, V. Jacobson, M. Vetterli, Receiver-driven Layered Multicast. *Proc. ACM SIGCOM '96*, August 1996, Stanford, CA.

[5] ITU-T Draft H.263: video coding for low bitrate communication. December, 1995.

[6] ITU-T Draft H.263+: video coding for low bitrate communication. September, 1997.

[7] ISO Draft in Progress MPEG4 Verification Model Version 5.1, July 1996.

[8] T. Braun, Internet protocols for multimedia communications, Parts I and II. *IEEE Multimedia*, July-September 1997, October-December 1997.

[9] A. Jacquin, H. Okada, P. Crouch, Content-adaptive postfiltering for very low bit rate video. *Proc. DCC '97*, March 1997, Snowbird, Utah.

[10] 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.

[11] Y.J. Chiu, T. Berger Perceptual rate control of video sequences. *Proc. Fourth Data Compression Industry Workshop*, March 1997, Snowbird, Utah.

[12] K.R. Rao, P. Yip, *Discrete Cosine Transform, Algorithms, Advantages, Applications*. Academic Press, 1990.

[13] H.G. Musmann, P. Pirsch, H.-J. Grallert, Advances in Picture Coding. *Proc. IEEE*, April 1985.

[14] B. Quinn, D. Shute, *Windows Sockets Network Programming*, Addison-Wesley, 1996.

[15] J. Boyce Private communication, 1997.