MIT/LCS/TR-542

VIDEO CODING AND THE APPLICATION LEVEL FRAMING PROTOCOL ARCHITECTURE

Andrew T. Heybey

June 1992

Video Coding and the Application Level Framing Protocol Architecture

by

Andrew Tyrrell Heybey

Submitted to the Department of Electrical Engineering and Computer Science In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology
June, 1991

Signature of Author ______________________________
Department of Electrical Engineering and Computer Science
May 10, 1991

Certified by ______________________________
Senior Research Scientist David D. Clark
Thesis Supervisor

Accepted by ______________________________
Professor Arthur C. Smith
Chairman, Committee on Graduate Students

Video Coding and the Application Level Protocol Architecture

by

Andrew Tyrrell Heybey

Submitted to the Department of Electrical Engineering and Computer Science
on May 10, 1991
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering and Computer Science

ABSTRACT

As networks and computers become faster, real time video transmission is expected to become common. Variable bit rate video coders will be used in order to take advantage of the statistical multiplexing gain and bandwidth efficiency of packet switched networks. Video streams have different service requirements from the traffic usually carried on computer networks. A new protocol architecture called Application Level Framing (ALF) has been proposed to allow efficient implementation of communications with diverse service requirements. ALF allows the application to control the way in which network errors are handled. This thesis studies the compatibility of three proposed video coding standards with an ALF protocol architecture. Each of the standards is found to be usable with varying degrees of effort. A set of design principles for video codes intended for use over an ALF protocol architecture is presented.

Thesis Supervisor: David D. Clark
Title: Senior Research Scientist

Acknowledgments

My thesis advisor Dave Clark helped me finally find and refine a topic and provided much guidance while writing the thesis.

Many thanks to my officemate Tim Shepard for his encouragement and sage advice on getting my thesis done in time.

Chuck Davin has listened to me for many hours and also proofread my thesis.

And finally many thanks to my wife Karen who proofread and who also loves, encourages, and puts up with me.

This research was supported by the Advanced Research Projects Agency of the Department of Defense, monitored by the National Aeronautics and Space Administration under Contract No. NAG2-582, and by a grant from the Corporation for National Research Initiatives.

This document was typeset with the LaTeX Document Preparation System.

Contents

Chapter 1

Introduction

Digitized video is an appealing use of computers and computer networks. As the bandwidth available on networks and the speed of computers increases, real time transmission of video between general purpose computers becomes a more and more realistic goal. However, there are limits to the available resources. There is not yet so much bandwidth available that the video can be sent without compression, and the service requirements of video are not satisfied by the usually available network protocols. Therefore new efficient protocol architectures must be employed for digital video transmission, along with video codes that take advantage of them.

1.1 Service Requirements of Video

The service requirements of video[1] differ in many ways from those of the traffic now typically carried on computer networks.

First, the bandwidth requirements of video are enormous. Uncompressed NTSC video requires approximately 50 megabits per second. Uncompressed HDTV requires much more. Both the transmission and storage of video suggest some form of com-

[1]Here "video" means a video sequence being displayed in real time. A video sequence being transmitted from one place of storage to another has exactly the same service requirements as any other data transfer.

pression. There is so much redundancy in most video sequences that even a relatively simple-minded compression scheme can result in a significant decrease in the space required. However, even when compressed, video without significant concessions in quality requires more sustained bandwidth than most other traffic.

Second, video is a real time service. If a computer is playing back a moving picture, the frames must be displayed at the correct frame rate without significant variation in the delay between each frame. This timing is lost during transmission across a statistically multiplexed computer network. Therefore, timing information must be encoded with the video data to allow the receiver to play back the video at the correct rate. There must also be buffering at the receiver to absorb variance in the delay across the network.

Finally, errors in a video stream require a different form of correction than errors in a normal data stream. Normal data transmission requires that all errors be corrected. The data must be delivered to the destination application in exactly the form and order they were sent by the source application. If data are lost en route, they must be retransmitted. In contrast, by the time a lost part of a video frame was retransmitted, it would usually be much too late to do any good. The proper reaction to a lost piece of video data varies with the way in which the video is coded. The correct reaction may be to do nothing, or to retransmit the state of that part of the screen at the current time, or some other more complex action. Hardly ever is the correct reaction to retransmit the exact data that were lost.

1.2 Constant Bit Rate Codecs

Many past video codecs (COder DECoders) have been designed to function in a leased-line fixed-bandwidth communication environment. Until very recently, a leased line was the only way to get guaranteed bandwidth and a small enough maximum delay. Constant bit rate (CBR) codecs must vary the picture quality to cope with changes in the information content of the video (or they must waste a large amount of bandwidth by reserving enough for the maximum burst rate required). They also assume that the only transmission errors will be bit errors with some relatively low probability, and that the delay across the communication link does not vary significantly. A variety of experimental and commercial codecs have been implemented under these assumptions. See [10], [8], and [9] for examples of CBR codecs. Chao and Johnston [4] built a constant bit rate codec that works over a packet-switched (ATM) network. Schooler and Casner [20] built a video teleconferencing system that runs over the Internet using a commercial CBR codec.

1.3 Variable Bit Rate Codecs

A constant bit rate codec has the big disadvantage that when a large burst of data must be sent (for example to accomplish a scene change) the extra bandwidth to perform the update simply does not exist. In contrast, a variable bit rate codec maintains a constant picture quality while producing a bursty data stream. A packet switched network can absorb such bursts without requiring all the bandwidth to be reserved. Packets switched networks are now becoming fast enough with a low enough delay to be able to carry video. Even the current Internet can support low bit rate video teleconferencing.

When using a packet switched network, video transmissions can exploit what is called the "statistical multiplexing gain." When many streams are multiplexed together on a statistically shared network, the total bandwidth required approaches the sum of the average rate of each video stream. For example, [22] states that when 64 variable rate video conference streams are multiplexed the total bandwidth required is about 64 times the average bandwidth. Since this is statistical multiplexing, there is still a possibility that the instantaneous bandwidth required will exceed this average (and the capacity of the network) so that packets will be lost. However, the probability of such an overload is no greater than the probability of a packet being lost due to transmission errors. Therefore, when many video streams share the same network link, they can each enjoy the constant quality of a VBR codec while the total capacity required is no more than the total average bit rate.

Variable bit rate codecs implemented to date mostly have used a packet-switched network in an ad-hoc way, bypassing the usual communication protocols in use on the network. Because of the memory bandwidth bottleneck described above, video cannot afford the processing and data-copying overhead of most protocols. The OSI seven layer reference model [11] strictly segregates the responsibility for reliability to the transport layer, away from the higher layers such as presentation and application. In this model, the transport layer does not know of the specific requirements of video, but delaying data while an earlier packet is retransmitted is completely out of place in a real time application like video. Late data are just as bad as lost data. Especially if video decoding is the bottleneck, the decoding process will never catch up once it gets behind because of delayed data. The correct reaction to a lost packet will vary with the video code used and the particular piece of data lost. The application that drives the codec is the only entity that knows the appropriate corrective action, and so it should be responsible for whatever level of reliability is needed.

Examples of previous VBR codecs include Verbiest and Pinnoo's codec [21] which is a piece of hardware that produces ATM packets directly. It can accept packet losses, but relies on the ATM network to preserve packet order. Other packet networks do not necessarily preserve order, and so to use this codec elsewhere would require modifications.

Moorhead, *et al* [14] built a VBR codec that uses arithmetic coding and both inter- and intraframe compression. They correct for packet losses rather crudely. The screen is periodically refreshed with an intracoded frame (every fourth or eighth frame). If a packet is lost, the screen freezes until the next intracoded frame arrives. There codec apparently has no way to only freeze the part of the screen affected or to otherwise disguise the error.

1.4 Application Level Framing

A new approach to network architecture has been proposed by Clark and Tennenhouse in [5] to aid in building protocols and applications that can run at speeds commensurate with new fast networks. The Application Level Framing architectural principles stress that the application knows best how to deal with reordering and losses in the network, and that memory bandwidth in the host is likely to be the bottleneck in the networks of the immediate future. Video makes a good testbed for the ALF ideas because of its unique service requirements as described above. The Application Data Unit is defined as the smallest piece of data that the application can accept out of order. The lower protocol layers are then responsible only for delivering complete ADUs to the application. Reliability and ordering are left up to the application itself.

1.5 Video Coding Standards

This thesis studies proposed video compression standards with the idea that it will be possible to buy off-the-shelf hardware to perform the compression and to interwork with many other sites around the world. There are many such standards being proposed both by international standards organizations and by private companies.

- The JPEG draft standard [12] proposed by a joint committee from the ISO and the CCITT is for compression of single images. While not designed with video in mind, it can certainly be used for video by compressing each frame individually. There already exists a hardware implementation [2].

- The MPEG draft standard [15] proposed by the ISO allows a variety of coding algorithms and provides for the audio accompaniment as well. A hardware implementation is planned by the same company that manufactures the JPEG chip.

- The CCITT's $p \times 64$ standard [3] is a predictive discrete cosine transform coder.
- CD-I [18] is a standard for recording interactive video and audio on compact disks. CD-I can only display full-motion video at less than full screen resolutions.
- DVI (digital video interactive) [19] is a proprietary scheme for compressing video for playback from a storage medium. It is designed to require much more work to compress than to uncompress, and so is unsuitable for real time transmission of video. It can be stored on any digital storage medium, though CD-ROM is a particularly attractive medium.

This thesis studies only the JPEG, MPEG, and $p \times 64$ video compression standards. CD-I does not support full frame full-motion video, and no meaningful information has been made public about DVI.

1.6 Other Work

Much work has been done on video codes and codecs of all types. Previous research on codes meant to be used over packet-switched networks has discussed the opportunities and challenges presented by such networks. However, studies of VBR codes and codecs have usually focused on the characteristics of one specific network. Little work has been devoted to protocols that can transmit video over networks with different packet sizes, loss rates, and service guarantees.

Wu and Lee discuss the transmission of video over different types of packet switched networks in [13]. While they discuss the ways in which different networks affect video transmission, they do not suggest any overall strategy that might be usable on all networks.

Brainard and Othmer [1] also discuss the effects of a general packet-switched network on video transmission. However, they assume that the network maintains packet order, thereby excluding a large class of networks from consideration.

1.7 Overview

Chapter 2 discusses Application Level Framing, the service requirements of video, and how well they mesh. Chapters 3, 4, and 5 summarize the three video compression standards and discuss the choice of ADU and design of an ALF application for each. Chapter 6 summarizes the characteristics of a video code that make it suitable for use with an ALF protocol, and names the compression standard that best meets those requirements.

Chapter 2

Application Level Framing and Video Coding

2.1 Application Level Framing

Application Level Framing (ALF) is a new set of network architectural principles set forth in [5]. ALF is motivated by several observations. Networks and processors are getting faster and faster, but memory bandwidth is not keeping pace. Also, while network bandwidth is increasing, network latency is bumping into the hard limit of the speed of light. In the interests of high performance it is therefore important to minimize the number of round trips across the network required to perform a communication and the number of times that a communicant must cycle its memory.

In the ALF model network protocol processing is divided into two parts: data manipulation and transfer control. Data manipulation is anything that requires reading or copying every bit of the bitstream, such as error detection, buffering, presentation, etc. Transfer control encompasses operations such as flow and congestion control, sending acknowledgments, detecting packet loss and reordering, etc. Because of the sheer volume involved, data manipulation is by far the more expensive step. By combining as many data manipulations as possible into a single integrated processing loop (a technique called *integrated layer processing* in [5]), the maximum processing efficiency is attained.

The strict layered division of responsibilities in traditional protocol architectures inhibits integrated layer processing. In general, each layer must complete its processing on the data before the next layer can begin, especially if each layer includes a multiplexing function. This structure can force serial processing and multiple data manipulation steps thereby inhibiting an efficient implementation.

In addition, this type of layering restricts the ability of an application that does not have typical service requirements to take advantage of the implementation efficiencies afforded (or demanded) by those requirements. The application is restricted to using those service classes provided by the lower layers, without the ability to bypass or improve upon them. A prime example of such an application is video, which wants neither a reliable bitstream nor an unreliable datagram service.

ALF has been decomposed into four layers in [7]:

NDU layer The NDU layer is responsible for the physical information exchange. The *network data unit* is the unit of data exchange in the underlying network (such as an IP packet or an ATM cell). The ADU layer uses the routing, congestion control, and resource allocation primitives provided by this layer to implement the higher level services.

ADU layer The *application data unit* is the smallest unit of data that the application can handle out of order. The ADU layer is responsible for transmitting ADUs across the network in terms of NDUs, and of notifying the application upon receipt or successful transmission of an ADU.

By allowing the application (the entity with the most complete knowledge of the structure of the information being transferred) to define the unit of information transfer, efficient use of host resources is encouraged. When an ADU arrives, the application performs all processing (error detection, presentation processing, and other application-specific processing) on it at once. This method limits the number of times that the data must be copied, the number of context switches that occur per ADU, and provides a great flexibility in the handling of ADUs. In addition, this efficiency is provided without tying the application to the characteristics of the physical network.

Application services layer This layer will provide common service classes (such as a reliable byte stream) based on the ADU layer. The application services layer exists only to avoid duplication of development effort. No application is required to use it.

Application layer This layer contains the application programs producing and consuming the data transmitted across the network. Applications use the ADU layer and/or the application services layer to create *flows* between themselves. A flow is a set of ADUs with the same source, destination, and service requirements. "The effort of setting up a flow is intended to be more or less what would be done for each packet in a connectionless network; the idea of a flow is that the results of this effort can be cached to good advantage." (From [5].)

The ALF application discussed in this thesis is the bit of code that goes between the video coder and the ADU layer. It is assumed that at least for the immediate future general purpose computers will not be fast enough to perform the video coding entirely in software. Therefore, the video coder will be a special purpose black box that produces or consumes a bitstream as specified in the relevant standard. The application in this case is responsible for converting that bitstream to ADUs and back again. If the coding could be performed in software, then there would not be such a strict line between the coder and the program that understands ADUs. There would just be a single application program that produces and consumes ADUs.

2.2 Video and Video Coding

Video is typically divided into one luminance and two chrominance components when it is digitally coded. This is in contrast to the typical computer color display, which uses red, green, and blue (RGB) components. Luminance and chrominance components are chosen because the chrominance components carry information less important to humans than the luminance, and can therefore be coded at a lower resolution. Codecs often sample each chrominance component at half the spatial resolution both vertically and horizontally. See Netravali and Haskell [17, Chapter 2], for more information.

Once the video has been divided into components and sampled, there are many kinds of coding that can be applied including transform coding, tree coding, vector quantization, and differential pulse code modulation. See Musmann, *et al* [16] or Netravali [17] for surveys of various coding types. Any type of coding can be either inter- or intraframe coded (also referred to as just intercoded or intracoded). An intracoded code relies only on the redundancy within a single video frame. Each frame is coded as an independent entity. An intercoded code also uses the temporal redundancy of video to perform compression. Such a code transmits the compressed differences from one frame to the next rather than complete frames. Each new frame thus depends on the contents of one (or more) previous frames. Interframe codes often further increase their compression ratios by performing *motion compensation.* In this case, the code includes motion vectors in the transmitted information. The part of the previous frame at spatial offset given by the motion vector is used as a predictor. Motion compensation allows moving objects in the video picture to be transmitted cheaply.

Interframe codes can provide a much higher level of compression than intraframe codes because of the high level of temporal redundancy inherent in video (especially a video with limited motion such as a teleconference). However, it is also much more difficult for an interframe codec to recover from transmission errors than an intraframe codec. Since newly transmitted frames depend on previous frames, an error will persist in the video display and in fact will propagate across the screen and grow if motion compensation is used. For this reason, interframe codecs (whether intended for circuit networks or packet switched networks) all incorporate some method to periodically refresh all or part of the screen using intraframe coding.

The video codes studied in this thesis all employ two dimensional discrete cosine transform (DCT) transform coding. A two dimensional transform transforms square blocks of pels into another domain before they are coded and transmitted. A useful transform concentrates the energy of the pels into a relatively small number of coefficients. The DCT produces a DC coefficient which is the average of all the pels in the block, and many AC coefficients that represent increasing spatial frequencies in the block. The coefficients of the higher frequencies can be transmitted at a lower precision than the DC and lower frequency coefficients. The DCT, while not an optimal transform, can be computed quickly. See [17, Chapter 5] for a complete discussion of transform coding.

2.3 Compatibility of ALF and a Video Code

In order to be carried by a network employing ALF protocol ideas, a video code must have several attributes. First, the bit stream produced by the coder must be easily divisible into ADUs that are neither too big nor too small. Second, the decoder must be able (with a small amount of help) to cope with the loss of an ADU. Finally, to mesh well with the ALF philosophy, the decoder should be able to process ADUs out of order. In determining the suitability of a code for ALF, the particular details of the coding used for the picture do not really matter that much (for example, DCT versus quadtree versus subband). What matters is how the bitstream is subdivided and framed.

2.3.1 Defining an ADU

An ADU that is too large suffers from several problems. If the error rate of the underlying network is non-negligible, it may not be possible to get a complete ADU through the network with any certainty. (At least not without implementing some form of reliable delivery of NDUs at the NDU layer, which goes against the ALF philosophy). Since the application receives only complete ADUs, the packetization delay of a large ADU may be unacceptable. Finally, the more data an ADU contains, the more difficult for the decoder to recover from its loss, and the greater the effect on the picture quality.

A too small ADU creates fewer problems. Depending on the amount of other information that the application must add to the compressed video data (such as timestamp, sequence number, CRC, etc.) a small ADU may require too much transmission overhead.

When considering the size of the ADU, it is tempting to tailor the size to the size of the NDU for practical reasons. It can be argued that an ADU which fits into the NDU is more efficient than one that is too large and must be fragmented and reassembled. Alternatively, if the network data unit is a fixed size, an ADU that is significantly smaller than the NDU uses network bandwidth inefficiently. This efficiency concern is a fallacy for several reasons. First, reassembly is not necessarily that expensive. There are a variety of implementations that can make it cheap relative to the other ADU processing that must be done. For example, reassembly does not have to mean copying the NDUs to construct an ADU that is contiguous in memory. Rather, the receiving application can be presented with a list of pointers to the NDUs making up the ADU. The fact that the ADU is not contiguous adds only a small amount of complexity to the processing loop that the receiving application will run over the data.

Second, the ADUs for the video codes discussed in this thesis will be of variable size. In all of the codes the compression ratio varies with the complexity of the scene, and the encoders and decoders operate on fixed-size areas of the screen. Therefore, if the bitstream is divided into pieces that the decoder can operate on naturally, the ADUs will be variable size. Sending fixed-size ADUs will either be impossible or add substantial complexity to the sending and receiving applications. Since the theoretical maximum size of an ADU is much larger than the typical size (the number of bytes in an ADU can vary by several orders of magnitude, depending on the parameters chosen for the compression), it is impractical to define an ADU that will never overflow the NDU. The ADU layer will have to be prepared to disassemble and reassemble ADUs in any case.

The size of the ADU versus the size of the NDU is much less important than whether the ADU is suited to the video code and ALF application. The same video protocol may be used over networks with radically different characteristics and NDU sizes, so designing the ADU with the NDU in mind is unwarranted.

2.3.2 Lost ADUs

Another requirement for the ADU is that the decoder (with the help of the ALF application) be able to cope with the loss of an ADU. The absolute minimum requirement is that the decoder not lose synchronization in the video stream and that the picture on the screen not fall apart. Beyond that minimum, there is a range of possible responses to ADU loss, which include the picture freezing until it can be refreshed, a part of the picture freezing, a noticeable loss of picture quality or at best an unnoticeable loss of picture quality. The feasibility of any particular correction depends on the details of the code and the network.

For any code, if the round trip delay between the sender and receiver is low enough, the receiver can ask the sender to retransmit the lost ADU. However, this action does not guarantee that the replacement will arrive in time. It too may be lost or arrive too late. The receiver must not rely on retransmission to fix the problem.

In the case of an intraframe code, the visual error will exist only until the display of the next frame. The error can be disguised by replacing the lost ADU with the same portion of the preceding frame. Alternatively the data from lost ADUs can be interpolated from the surrounding ADUs in the same frame. The feasibility of interpolation depends on the code used. For the transform codes studied in this thesis (where an ADU might be a block of 16 by 16 pels) interpolation is not likely to be a successful strategy.

In the case of an interframe code that incorporates motion compensation, an error due to a lost ADU will persist and propagate across the screen. A lost ADU means that a part of the frame to which the ADU belongs will be incorrect. Because the succeeding frame is sent using the current frame as a predictor, the error will persist from frame to frame (and probably get worse as differences are applied to an increasingly incorrect predictor). Since later ADUs are transmitted using motion compensation, the error will propagate spatially. To correct the error, the transmitter must transmit a correction large enough to cover the extent of the damage.

If the coder corrects errors by transmitting extra data, the sender runs the risk of falling over a precipice into an unwelcome operating region. Depending on the amount of extra data transmitted and the probability of losing an ADU, the extra bandwidth consumed to fix errors could grow extremely large. The more corrections that must be sent, the more bits are transmitted. The more bits that are transmitted, the greater the chance of a lost ADU for a given network error probability. The more errors, the more corrections that must be transmitted and the greater the bitrate required. The extra bandwidth required may grow arbitrarily large, depending on the design and/or stupidity of the coder.[1] Even with a relatively intelligent sender, the code could quickly degenerate to sending every frame intraframe coded. Once this poor performance point is reached, the encoder may never recover.

[1]Imagine that through some burst of errors, every ADU in a frame is lost. Further imagine that the network delay and motion compensation possible are such that the sender believes it must refresh the entire screen for any single lost ADU. If the sender does not notice that it has already refreshed the screen for the first lost ADU, it will proceed to refresh the screen many times, once for each lost ADU. This burst of data will have a high probability of losing several ADUs.

One alternative to the brute-force "retransmit everything that might have been damaged" approach is to keep track of the motion compensation used in each block in prior frames, and to transmit the appropriate differences to correct the error. Wada proposes such a scheme in [23]. When an ADU is lost, the receiver displays the data from the previous frame and sends back a negative acknowledgment for the block to the sender. The sender then applies that same "fix" to its frame store and continues to code the video stream using the incorrect picture as a predictor. These differences will correct the error when they arrive without having to either retransmit any data or refresh the screen. The complexity of such a scheme must be balanced against any savings in bandwidth due to the smaller number of intracoded blocks sent. The sender must maintain a store of old frames at least as long as the maximum expected round trip transmission delay. In addition, when it learns of a lost ADU, the sender must propagate the error through its local frame stores by executing the decoding algorithm on the bitstream with the missing ADU. All of this requires a significant amount of memory and CPU cycles.

Another approach to dealing with lost ADUs is to use a code that tolerates them. This approach is very promising, especially with a network that allows different flows to have different priorities. The ADUs carrying low resolution information are sent on a flow with a higher priority than those carrying information to fill in the details. If ADUs are lost due to congestion, the picture just becomes fuzzy (possibly unnoticeably so) for a little while. Subband codes are very well suited to prioritized transmission. Darragh and Baker discuss a subband code and its sensitivity to errors in [6]. They note that "missing non-baseband packets have minimal effect on the subjective image quality." They replace missing baseband packets with the interpolation of the pels above and below the missing ones (their packets consists of a single line of pels), and state that such losses produces a minimal effect on the picture quality.

Even if prioritized transmission eliminates the loss of ADUs with high priority, any ADU may still be lost due to transmission errors. The receiver must still be able to deal with such losses by one of the other methods described above. Of the codes studied in this thesis only JPEG could take this approach. The multiple priority code violates the constant quality promise made for variable bit rate codes. However, it does so on a statistical basis (when bursts happen to coincide the in network), not because of a fixed limit on transmission bandwidth.

Finally, note that in order to perform even the minimal recovery from lost ADUs for a black box decoder, the ALF application has to know at least a little bit about the structure of the bitstream. At the very least, the application has to know how to construct a place-holder ADU that does not violate any invariants that the decoder expects. The more complex those invariants, the more the receiving application has to know about the code and the more it has to understand the contents of the ADUs.

2.3.3 Out of Order ADUs

Out-of-order ADUs are very closely related to, but slightly different from lost ADUs. A lost ADU causes all the following ADUs to be "out of order" with respect to the lost ADU. However, if the decoder cannot tolerate out-of-order ADUs the receiving application can (assuming that it knows enough about the code) construct an artificial place-holder ADU to fill in for the lost one. The decoder then does not see the ADUs as being out of order. If the decoder can deal with out-of-order ADUs, then no such place holder is required. The decoder can immediately be fed the subsequent ADUs, and the missing ADU can be sent to the decoder when and if it arrives.

The need for out-of-order processing arises for two reasons. First, ADUs may be reordered by the network. Second, an ADU which is lost and then successfully retransmitted will arrive out of order. In either case, out-of-order processing makes it possible to keep the decoder busy in the face of lost and reordered ADUs. If the decoder is the bottleneck in the system, it must be able to process the ADUs as they arrive, without waiting until they are put back into sequence. Otherwise the decoder will fall farther and farther behind for every ADU that is reordered for any reason.

While not strictly necessary to be usable in an ALF protocol architecture, the decoder should be able to process ADUs out of order as much as possible. The application can reorder the ADUs if necessary, but if the choice of the ADU and characteristics of the code do not allow at least some out-of-order processing, one of the strong points of the ALF architecture is eliminated.

2.3.4 Header Data

Replicating Header Data for Reliability

There is an additional wrinkle to choosing an ADU for each of the compression schemes. The ADU will, in general, be as small as is practical. However, all the algorithms have additional header data for the larger units of pictures, groups of blocks, etc. These data can not be lost without affecting at least the entire frame. Therefore, whichever unit is chosen as ADU, if the header data for one of the larger units are lost the decoder is in trouble. Even though subsequent ADUs appear to be independent, they implicitly need the information that was lost. Even worse, it is possible to lose the data without noticing the loss, as much of this information is an optional part of the bitstream.

It is possible to fix the problem by including the needed information in every ADU. However, depending on the amount of data (indices would probably be small enough to duplicate; entire Huffman code tables would not), this strategy may impose an unacceptable duplication of data, particularly in the case of large tables that do not change often.

To avoid superfluous duplication of large amounts of header data, the sender can create an additional flow for the table updates. In this flow, each ADU is a table update. When a new table is produced by the encoder, it is transmitted on the table update flow. Each table is given a version number, and the ADUs in the video flow carry the version number(s) of the table(s) they need for decoding. To avoid decoding ADUs incorrectly, the table update flow should be reliable. The reliability can be achieved in one of two ways. The first is the more traditional. The receiver sends an acknowledgment upon receipt of each table update. If an acknowledgment is not received within a certain amount of time, the sender retransmits the table. This retransmission strategy is well understood. However, it might not provide quick enough recovery from lost tables if the sender does not time out soon enough. An alternative is for the arrival at the receiver of a video ADU containing a table version number not yet seen to prompt the sending of a negative acknowledgment to the transmitter. This strategy can either be used by itself or as an addition to the positive acknowledgment strategy.

Using a separate reliable flow to transmit the table updates has the problem that there is no guaranteed bound on the time it will take to get a table update to the receiver. Unlike the loss of a regular ADU, the loss of a table update may force the decoder to stop or to produce a very poor picture. A different strategy does not require a separate flow or retransmissions with their accompanying uncertainty. When a new table appears in the bitstream, it is sent as part of every video ADU until an acknowledgment is received for that table. Each such unit of reliably-transmitted data must have its own sequence number (or other means of unique identification). When the receiver acknowledges the sequence number of each unit of reliably-transmitted data, the sender stops including the data with each ADU. This method of transmitting data that must arrive reliably has the property that no ADU will ever arrive at the receiver without all the information needed to decode it, at the cost of the extra bandwidth for the needlessly duplicated tables. It also requires the video ADUs and receiving application to be more complex to detect and process the optional tables in the ADU.

Eliminating Redundant Header Data

While dividing the compressed video stream into ADUs requires the complexity of duplicating the signalling information as described above, it also has advantages. The part of the bitstream contained in each ADU is framed by the ADU itself. Therefore, any framing information that delineated that part of the bitstream can be omitted. The sending application can remove the framing information, and the receiver can replace the framing in the stream being sent to the decoder.

2.4 Design of an ALF Video Application

The obvious design of a system using ALF to transmit encoded video over a packet-switched network consists of three black boxes at each end of the network. The three boxes at the transmitter are the encoder, the ALF application, and the ADU layer. The encoder takes a video signal as input and produces a bit stream in the format specified by the implemented video coding standard. The application takes the encoded bitstream as input and divides it up into ADUs adding timestamps and sequence numbers as necessary. Finally, the ADU layer takes ADUs and sends them over the network to their destination.

At the receiver the boxes are the ADU layer, the ALF application, and the decoder. The ADU layer receives network data units from the network, and produces ADUs for the application. The application receives the ADUs and reconstructs the bitstream according to the coding standard used, and then feeds the bitstream to the decoder which produces a video signal.

The problem with this obvious design of the system is that requiring the encoder and decoder to be black boxes eliminates much of the advantage of the ALF protocol architecture. All of the codes proposed for standardization do not make any formal provision for decoding their bitstreams out of order. (However, as discussed in this thesis, all of the standards could be processed out-of-order if the coded stream is divided into appropriate units and the decoder is prepared to deal with them.) The C-Cube JPEG chip, for example, can decode the bitstream only in order. Therefore, if an ADU is received out-of-order, the decoding process must grind to a halt. The only alternative is to immediately treat the out-of-order ADU as lost.

To achieve the maximum benefit from the ALF principles, the receiving ALF application must be able to send ADUs to the decoder out-of-order. This implies that the communication between the decoder and the application must be more than just a bitstream. The decoder (whether it is implemented in software or hardware) must be able to communicate with the application to find out where to put each ADU.

2.5 Multiple Video Streams

The proposed video compression standards studied in this thesis all implicitly assume that the communications channel to be employed is a fixed-bandwidth point-to-point circuit. From this assumption it follows that the decoder will decode one stream at a time. In contrast, a statistically multiplexed network encourages the transmission of many simultaneous streams. Even though the decoder may be a black box designed to decode a single stream, the receiver would like to multiplex the decoder for many different video streams. To use the same decoder for many video streams, the receiving application must be able to quickly reload any state in the decoder as each ADU is processed. If the decoder expects to receive only one video stream, that state may be hidden or difficult to change. In this case the application must use a separate decoder for each video stream. The decoder must also operate quickly enough to decode multiple streams in real time.

Chapter 3

JPEG

3.1 Description of Algorithm

JPEG [12] is an image compression standard, meaning that it performs only intraframe coding. However, it can be used for video by coding each frame individually if the encoding and decoding are performed fast enough. Of course the compression achieved will generally be less than for algorithms that perform interframe coding.

The JPEG draft standard encompasses both a discrete cosine transform (DCT) algorithm and a lossless differential pulse code modulation (DPCM) algorithm. The DCT algorithms can entropy code the coefficients using Huffman coding or arithmetic coding. The DCT coefficients can be transmitted sequentially, progressively in order of increasing spatial frequency, or progressively with increasing precision. Both the DCT and the DPCM algorithms can transmit the image hierarchically. In this mode, the image is first coded and transmitted (using either DCT or DPCM) at a lower resolution and/or lower quality than desired for the final image. This low-quality image is used as a predictor and differences from it to a higher-quality image are computed and transmitted. This process is repeated until the final image is obtained. It would be possible to use the hierarchical mode in a network that supports different priorities for transmitted data. The initial frame could be transmitted at the highest priority, and subsequent frames at lower priorities. If the network became congested, it would drop lower-priority packets first, which would result in an overall drop in picture quality rather than a glitch on the screen.

The most likely candidate for video would be the DCT algorithm. The DPCM algorithm is neither fast enough nor necessary for a moving picture. DPCM does not achieve as great a compression ratio, and losslessness is even less important for video than for still images. The standard claims to achieve very good quality images with DCT coding, so there is no need to use the extra bandwidth to transmit DPCM images for video.

A compressed JPEG image is constructed of the following parts. In the bitstream, signalling information is identified by the hexidecimal value $FF followed by a marker byte identifying which information follows. If $FF appears in the coded data, the encoder must insert a zero byte after it to indicate that it is not a special code.

image The single image is the largest unit of information defined by JPEG. Each video frame will be transmitted as a JPEG image. An image consists of one or more components (up to 255 components per image). The JPEG draft standard does not specify that any particular set of components be used. A common set of components used for video is one luminance component and two chrominance components, with the chrominance components sampled at one-half the spatial resolution of the luminance component.

frame A JPEG image contains one or more frames. Images contain more than one frame only if they are coded using hierarchical mode. Note that the different JPEG "frames" of an image are all part of the same video frame. Unless otherwise specified, the word "frame" in this chapter refers to the usual video frame, not the JPEG frame. The signalling information for each frame includes the type of coding used (DCT or DPCM, Huffman or arithmetic coding, sequential or progressive, etc.), the number of lines and pels per line, the data precision, the number of components, the resolution of each component and which quantization table to use for each component. There may also be Huffman tables, quantization tables, arithmetic coding tables, and definition of the restart interval before the start of each frame in the image. If these tables are included in the bitstream, they stay in effect for later images as well as the current image.

scan A frame consists of one or more scans through the image data. Each scan can contain one to four components of the image. A scan may be interleaved or non-interleaved. A non-interleaved scan contains one component of the image. An interleaved scan contains more than one component. The signalling information for each scan identifies the components of the image included in the scan and which Huffman tables to use for each component. New Huffman and quantization tables may also appear in the bitstream before the start of each scan (as described above for the JPEG frame).

minimum data unit Each scan consists of a sequence of minimum data units (MDUs). For non-interleaved scans (which contain only one component), the MDU is either a single eight by eight block (for DCT) or a single sample (for DPCM). For interleaved scans, an MDU is at least one block or sample from each component. If the components have different resolutions, the number of blocks or samples from each component is such that the total physical screen area covered for each component is equal. For example, if the luminance component has twice the horizontal and vertical resolution as the two chrominance components (as is a common case for video), the MDU will consist of four blocks from the luminance component and one block from each of the chrominance components.

block The eight by eight block is the basic unit of DCT compression. The DC coefficient is DPCM coded from that of the previous block of the same component. The AC coefficients are either Huffman or arithmetically coded in a way that efficiently encodes runs of zero coefficients. The end of each block is marked by a special marker. If the block ends before all 64 coefficients have been transmitted, the rest of the coefficients are implicitly zero.

restart interval A scan may be broken up into restart intervals. At the beginning of every restart interval, the predictor of the DC coefficient of the DCT is set to zero. The restart interval *must* be defined at the beginning of the image. Each restart interval contains a fixed number of MDUs, except for the last one in a scan which may be shorter. Restart intervals are identified in the stream using a modulo-eight counter.

3.2 Choice of ADU

The smallest unit that could be used as an ADU is the restart interval. At the beginning of every restart interval, the predictor for the DC coefficient is set to zero, so decoding does not depend on any other ADUs in the frame. JPEG does not include any interframe coding, so each restart interval is completely independent (except for any redefined quantization or Huffman code tables). If transmitting video, the ALF application would have to add some more identification (such as sequence number and/or frame number and/or timestamp) to each ADU so that it could be placed in the correct position in the appropriate video frame. Since new Huffman and quantization tables can be defined before any frame or scan, any such new tables must be transmitted reliably as discussed in Section 2.3.4. They are too large to be included in every ADU transmitted.

The various signalling information included with the scan and the frame is also needed to decode the ADUs. However, this information will not change from ADU to ADU, and in fact will probably not change from video frame to video frame. Whether to send this information with every ADU or to only send it when it changes (like the tables), depends on the size of the ADU.

3.3 Design of an ALF Application

There are several issues to consider in the design of ALF applications to transmit and receive JPEG. The first is how to deal with the transmission of the various tables that may be included in the bit stream. These tables are large enough (several hundred bytes) that it is impractical to transmit the applicable ones with every ADU. Instead, as discussed in Section 2.3.4, the tables will be transmitted in some reliable manner when they change.

The receiver must be careful to redefine the tables only at the right time. Old tables must not be replaced too soon. If ADUs are reordered, then an ADU needing a new table may arrive before all the ADUs using the old table have arrived and been decoded. In this case, the old table must be retained until it is not needed any more. The old table must be kept until the last ADU that might need it has become too old (where "too old" is defined below). If changing the tables is expensive, then all ADUs before a table change must be decoded before any ADUs following the table change.

The job of the transmitting application is relatively simple, and does not depend much on the particular encoder used. It packages each restart interval produced by the encoder into an ADU. Each ADU gets a header containing all the signalling information as described above, a timestamp and a sequence number. To provide the most information to the receiver, the timestamp should be the midpoint of the time interval during which this restart interval was displayed on the screen in the source video. The sequence number is an integer which is incremented for each ADU,allowing the receiver to order the ADUs (if necessary) and to detect duplicate and lost ADUs. If the encoder and decoder support hierarchical mode and the underlying network supports priorities, each ADU is also given a priority corresponding to its JPEG frame number within an image.

The receiving application is somewhat more complex. It must discard ADUs which arrive too late, give any new tables or other signalling information to the decoder at the correct time, deal with lost and duplicate ADUs, perform any reordering that the particular decoder requires, and send the restart intervals to the decoder. The implementation of all of these functions (except for discarding ADUs which arrive too late) depends to a large extent on the implementation of the decoder. As an example, the next section describes the design of an ALF application to work with a commercially available JPEG encoder/decoder chip.

3.3.1 Design for the C-Cube Chip

The C-Cube Microsystems CL550 JPEG Image Compression Processor [2] implements the baseline JPEG draft standard. The baseline standard is the sequential DCT algorithm with Huffman coding of the coefficients. It also limits the size of the image, the pel precision, and the number of Huffman tables used. The C-Cube chip has a "pixel interface" on one side and a "host interface" on the other. The pixel interface can produce or consume raster-scan pixels as video (including the generation of, or synchronization to, horizontal and vertical sync pulses). The pixel interface requires some external RAM as a line buffer in order to convert from raster scan video to the eight-by-eight blocks needed for the DCT and back again. The chip itself does not interpret any marker codes in the bitstream except for the restart interval code nor does it produce any marker codes except for the restart interval. External software must add on the rest of the framing during compression and strip it off during decompression. The quantization and Huffman code tables are programmable.

The sending application for the C-Cube chip must have an external source for the signalling information defined in the JPEG bitstream since the chip produces only restart interval marker codes and no other framing or signalling information. Besides having to produce the signalling information, the sender works as described above in the general description.

Unfortunately, if the C-Cube JPEG chip is used as the decoder, the advantages of ALF are largely moot. The chip does not provide for any way to process the data out of order. There are no addresses in restart intervals (they are addressed by their position in the data stream), and the chip does not have any way to specify where a restart interval belongs on the screen. The receiving application is limited to reconstructing the bitstream verbatim. If an ADU is lost the receiving application must put a placeholder in the bitstream so that the chip does not get confused, but that is the extent of the permissible variation. The other signalling information in the standard is not needed by the C-Cube chip. A number of registers must be initialized with the size of the frame etc., but they need not be touched from frame to frame unless the signalling data indicate a change. To elaborate, the receiving application must perform the following tasks to prepare arrived ADUs for the C-Cube chip:

1. Determine if the ADU is too late. "Too late" means that the timestamp in the ADU plus the decoding delay is less than the timestamp of the oldest ADU not yet displayed on the screen. In other words, each ADU must arrive with enough time to be decoded before it must be displayed. If an ADU arrives too late it is discarded.

2. Reorder the ADUs in left-to-right, top-to-bottom order. Since there are no addresses encoded in the restart intervals (and no other way to communicate an address to the decoder), the chip must receive them in the correct order with no gaps. The ADU will also contain a timestamp and/or frame number so that ADUs from different video frames can be differentiated (this information is, of course, not included in the standard, since JPEG concerns itself with single images only).

3. Install any new Huffman or quantizer tables that arrive and send any acknowledgments that are required for the tables. The tables must be installed in the chip at the correct temporal location in the stream so that each ADU is decoded using the correct tables.

4. Deal with lost ADUs. If the receiver has not received an ADU by the time that

it must be sent off to the decoder, that ADU is considered lost. Because JPEG uses only intraframe coding, minimal action is required when an ADU is lost. The disruption to the current frame should be minimized, but the error will be overwritten with the next frame anyway.

There are several ways to design the application so that it can hide the effects of a lost ADU. One design is to keep track of each frame as a unit. The application sets a timer for the time when the next frame should be sent to the chip. When the timer goes off, the application checks to see if all the ADUs in the frame have arrived. If there are any gaps, it fills them with the corresponding ADU from the previous frame or an artificial all-grey ADU. Finally, all the ADUs of the frame are placed on a queue to be sent to the chip. There must be a separate task or interrupt handler whose sole responsibility is to feed the JPEG chip from that queue.

Note that if the Huffman or quantizer tables have changed since the previous frame, an old ADU cannot simply be substituted for the lost one. If the tables have changed, an artificial ADU must be used. The all-grey ADU will also change with different Huffman and quantizer code tables. The application could substitute a restart interval of all zero coefficients (since the all-zero ADU will not change, no matter what the Huffman or quantization tables). This substitution would not look very good but would satisfy the minimum requirement of not letting the decoder get out of synchronization.

Reassembling complete frames before sending them to the decoder allows the least leeway in arrival for the ADUs. An alternative is to operate at a finer granularity. Set a timer for the time that each restart interval should be sent to the chip. When the timer goes off, if the ADU has arrived, put it (and subsequent ADUs in sequence that have arrived) on the queue to be sent to the chip. If the ADU has not arrived, substitute the same ADU from the previous frame or an artificial one. This involves setting more timers, but it allows the

ADUs to be later (and more out-of-order) without the image suffering at all.

Another alternative is a combination of the last two. Queue complete frames to be sent to the chip (with placeholders for missing ADUs), but reach in and insert any ADUs that arrive in time. The application must be careful to know just how much of the frame has been sent to the chip to avoid completely destroying an ADU. It also requires a more complex buffer-management scheme, since ADUs are variable-sized. There must be enough room in the buffer containing the placeholder, or the ADUs must be threaded on a list. Otherwise it will be impossible to insert the late ADU in the queue.

The final approach is to request the retransmission of the lost ADU from the sender. This approach is only feasible if the round trip delay between the sender and receiver is very short and the decoder is not the bottleneck in the system. Since the ADUs must be decoded in order, all decoding must be held up until the replacement arrives. Also, the application must still be prepared to cope (by using one of the strategies discussed above) with any retransmissions that do not arrive in time.

5. Discard duplicate ADUs. These are simple to detect. In the process of reordering the ADUs, if the application finds an ADU in the position where the newly-arrived ADU belongs, the newly-arrived ADU is discarded. If older ADU has already been sent to the decoder, then the duplicate should have been discarded by the "too late" detector.

3.3.2 Modifying the C-Cube Chip

The major limitation of the C-Cube JPEG chip is the requirement that ADUs be decoded in order. As discussed above, there is nothing about the code that enforces this requirement. The problem is that the ADUs contain only the most primitive address in their header (a modulo eight counter), and the C-Cube chip does not provide any way to tell it which restart interval belongs where in the image. For the purposes of an ALF protocol, a better decoder design would include the ability to specify an address with each ADU passed to the decoder. At a minimum, the address would specify the location in the image that the ADU covers. At best, the address could also specify to which image the ADU belongs.

The ability to specify an address with each ADU comes with some tradeoffs. The C-Cube chip does not require a frame store to operate. It needs a line buffer to convert the image from raster scan to eight-by-eight blocks and back again, but it needs only a few tens of lines of buffering for this purpose. A decoder that could handle ADUs within the current frame out of order would need a single frame store. A decoder that could handle ADUs out of order from different frames would need multiple frame stores and a method to switch between them. Depending on the characteristics of the intervening network, and whether the decoder is the bottleneck in the communication, the ability to keep the decoder's pipe full is an important enough advantage to outweigh the added cost of a frame store (or several frame stores).

To add the capability to randomly address ADUs, the C-Cube chip needs some extra hardware. First, the pixel interface must be changed to address at least one frame store rather than the current line buffer. The current line buffer is effectively a slice of a frame store, so this change requires only a widening of the the line buffer address. The ubiquitous megapel display requires 20 bits of address (though a megapel is overkill for NTSC video), plus additional bits if the framestore can hold more than one frame. Twenty-four bits of address is a minimum and more are desirable.

Second, there must be a method to pass the decoder an address indicating the frame and the location within the frame of each restart interval. The simplest way to accomplish this is to use that address to initialize an external address register addressing the framestore. The application must compute the address in the frame store of the beginning of each ADU. The current complex address counting that the

C-Cube chip performs on its pixel bus would be ignored in the new scheme, since the conversion from eight-by-eight blocks to raster scan is no longer needed. The chip (with the external pel address register) would just lay down the blocks in the frame store in order, starting at the address passed by the application.

The simplest way to add an address register that the application can initialize is to install an external address register and ignore the pixel bus addresses generated by the C-Cube chip. This register must also be a counter in order to generate all the addresses for all the pels of each ADU. In order to put the pixels in the correct place in the frame store (assuming that the frame store is read in the usual raster-scan order), the address register/counter must count so as to step through the frame store in eight by eight blocks. The application must initialize the address register for each ADU fed to the chip.

The biggest problem is loading the external address register with the starting address of each ADU exactly when the pels from that decoded ADU start to emerge from the C-Cube chip. The application does not care about the ADU anymore at this point. It will have handed off the ADU to be fed to the C-Cube chip by some other task. In order to load the address register at the correct time, there must be a FIFO between it and the application. When the application queues an ADU for the C-Cube chip, it also puts the address of the ADU in the FIFO. The trick is now to decide when to load the address register from the FIFO. The pel in the upper lefthand corner of a row of blocks is accessed first in both raster scan and block order. If the restart interval (and hence ADU) is made to be one row of blocks, then the address register should be loaded every time the C-Cube chip emits a zero address on the pixel bus.

If the ADU cannot be one row of blocks, then a more error-prone scheme can be used. The address register is loaded from the FIFO every n pixels, where n is the number of pels in an ADU. As long as the ADUs contain a constant number of pels, and the counter that pokes the address register every n pels does not get

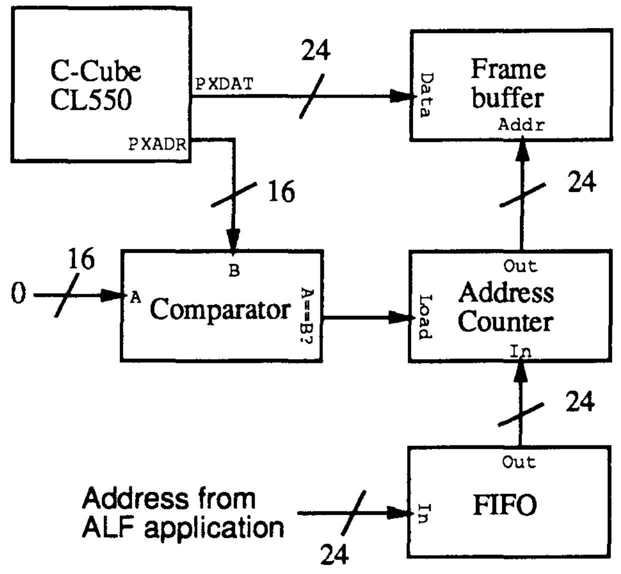

Figure 3.1: Adding an external address counter

out of phase, this scheme will work. However, if the counter does manage to get out of phase with the decoder chip, the scheme falls apart. Alternatively, the pixel bus address put out by the C-Cube chip could be compared to a list of addresses, each of which is known to occur at the start of an ADU. However, this scheme is just as error prone as counting the number of pels per ADU. If an ADU is not an entire row, then the address signifying the start of an ADU varies from scan to scan because of the pattern that the C-Cube chips uses to generate addresses. The only pixel bus addresses that always refer to the same physical pel on the screen are those that are the same whether the screen is scanned block-by-block or line-by-line (for example the pel in the upper left corner of a row of blocks, which is address zero).

A block diagram of the external address register and associated FIFO, counters, and support circuitry is in Figure 3.1. The modified hardware does not require any modifications to the ADU. The calculation of a frame store address for each ADU does not require any more information than the application already has to reorder the ADUs.

Now that the decoder can take a frame store address for each ADU, the ALF receiver application has much more latitude in the way it can pass ADUs to the decoder. The application must still discard ADUs that are too late and deal appropriately with new tables, but the operation in the presence of lost or reordered ADUs is different. Another external circuit will scan and convert each frame store to one video frame in a round robin order. As each ADU arrives, its frame number and frame store address is computed. The ADU is immediately queued for output to the decoder chip if its frame is currently occupying one of the frame stores and if there is enough time for the ADU to be decoded before its frame is displayed. If it is too late, the ADU is discarded. If its frame is so far in the future that there is not yet a frame store for it, the ADU is put on a queue for "too early" ADUs. There is also still some need for reordering ADUs. If an ADU arrives radically out of order, it may be too late if it is placed at the end of the queue, but in time if placed at the front of the queue. Therefore the application should enqueue the ADUs for decoding based on a rough estimate of how late they are. This is much easier than maintaining a queue ordered by sequence number since merely deciding whether to place the ADU at the head or the tail will suffice.

Lost ADUs are now treated somewhat differently than with the unmodified C-Cube chip. When the display time for each video frame arrives, some of the ADUs for the frame may not have arrived because they were lost or delayed. The application can take one of several approaches to handle lost ADUs:

1. The simplest choice is to do nothing. This means that the framestore will contain garbage where the lost ADU should be. In practice, this garbage will be the same portion of the frame from several frame times ago. This approach will be visually ugly if the picture has changed significantly during the last few frames, but requires no bookkeeping or extra processing.

2. If there is enough framestore bandwidth available, the application can initialize each frame store to some value before beginning to decode ADUs into it. Any

missing ADUs will then be grey (or whatever other color chosen) areas on the screen. This approach requires that the application be able to write directly into the frame store (note that neither the existing C-Cube chip nor the proposed modification require this capability). It also assures that any lost ADUs will be visually noticeable.

3. The application can keep track of which ADUs have not arrived, and substitute some other placeholder ADU for the missing one, as described above in the section about the unmodified C-Cube chip. Substituting a constant ADU will have the same visual effect as initializing the frame store to some value. However, the application then does not need to directly write to the frame store. It will have to keep track of which ADUs have been lost, and must start sending the placeholder ADUs to the decoder early enough. For example, if calamity strikes and half the ADUs are lost, the application must start sending the substitute ADUs to the decoder half a frame time in advance, to be sure that the decoder can complete the frame before it must be displayed.

4. Another approach is similar to the previous one, but instead of substituting a constant ADU for the missing ones, the application substitutes the same ADU from the previous frame. This approach requires the most bookkeeping on the part of the application (it must both keep track of which ADUs have not arrived and save the last n frames worth of ADUs), but has the potential for producing the best visual results. It also does not require that the application be able to directly write into the frame store. If the same ADU was lost in the previous frame, the application would look for the frame before that, and so on. If the same ADU was lost in the preceding n frames, the application could just omit that ADU from this frame. This case should be rare and produces results no worse than the first approach discussed.

5. The application can also request retransmission of the lost ADU with similar

conditions and caveats as discussed above for the unmodified chip. However, since the ADUs no longer need to be decoded in order, retransmission is more feasible. The application needs to decide when to ask for a retransmission. At least one round-trip time (plus the decoding time for each missing ADU) before the frame is displayed, the application must send a retransmission request for all ADUs that have not yet arrived. If they arrive in time, the application feeds them to the decoder and all is well. Otherwise, it must employ one of the other lost ADU recovery methods.

The ability to decode out-of-order ADUs also alters the treatment of duplicate ADUs. When using the C-Cube chip by itself, duplicate ADUs were detected and discarded by the code that reordered the ADUs before sending them to the decoder. Since there is no need to reorder ADUs any more, the application loses the ability to detect duplicates "for free." However, decoding the same ADU twice does not affect the video picture; it merely wastes resources. As long as the ADU duplication rate is relatively small, is is more effective to waste the resources required to decode the occasional ADU twice than to expend the resources to check every ADU for duplication.

3.4 Multiple Streams

The C-Cube chip (with an external address register) works very well at decoding multiple video streams *if* every stream uses the same picture components, horizontal and vertical size, and Huffman and quantization tables. The ALF application just computes the correct frame store address for each ADU and puts the ADU with its address into the queue to be sent to the chip. Since each ADU is entirely independent of all others (including those from other video streams), the order in which they are decoded is irrelevant as long as the output is deposited in the correct place in the frame store. If the parameters such as resolution and number of components are different among the various video streams, the ability to multiplex the C-Cube chip depends on how fast its internal state registers can be changed.

Chapter 4

MPEG

4.1 Description of Algorithm

MPEG [15] is intended to code both audio and video. This thesis studies only the video code. The video part of MPEG codes sequences of frames. Each frame is composed of one luminance and two chrominance components. The chrominance components are sampled at half the rate (both vertically and horizontally) of the luminance component. MPEG uses a discrete cosine transform code, and both intra- and intercoded frames.

MPEG-coded video is divided into the following parts:

video sequence A video sequence is one or more groups of pictures that have the same parameters (such as picture rate, picture width, and picture height). The header of a video sequence includes all this information as well as optional quantizer matrices and optional extension data and user data.

group of pictures A group of pictures (GOP) is a set of one or more pictures that can be decoded without reference to other pictures in the video sequence. The group of pictures is the unit of random access in a video stream. A GOP must begin with an intracoded picture, though intercoded pictures may exist in the

GOP that depend on other GOPs. If the video sequence is not played in order, pictures that depend on earlier GOPs are not displayed. The GOP header contains an SMPTE (Society of Motion Picture and Television Engineers) time stamp and flags which indicate whether there are any pictures in the GOP that depend on pictures in other GOPs. The timestamp gives the relative time between the picture in this GOP with temporal reference zero and the picture in the preceding GOP with temporal reference zero. (The temporal reference number of a picture is described below.)

picture There are four types of pictures. An *intra-coded* picture (I-picture) can be decoded without any other information. A *predictive-coded* picture (P-picture) uses motion compensated prediction from an earlier picture. A *bidirectionally predictive-coded* picture (B-picture) uses motion compensated prediction from a past and/or future picture. A *dc coded picture* (D-picture) contains only the DCT DC coefficient information.[1]

The picture header contains a temporal reference number, the picture type, an optional time stamp, and some information about how to decode any motion compensation vectors in the picture. The temporal reference number orders the pictures in a GOP in time, from zero through n where n is the number of pictures in the GOP. The pictures in the GOP may be in the bitstream out of order (for example B-pictures are transmitted after the picture used for prediction). The timestamp is in units of 1/90000 second, and may be relative to the timestamp of the preceding picture or an absolute value.

slice A picture consists of one or more slices. Each slice consists of one or more macroblocks of the picture in order. Slices do not overlap. The slice header contains the vertical position of the slice and the quantizer scale used for the slice. The vertical position is the macroblock row of the first macroblock in

[1]This thesis will not consider D-pictures, since the standard does not discuss their use.

the slice. The quantizer scale is a number from one to 31 that indicates which quantizer level to use when decoding the DCT coefficients.

macroblock A macroblock contains a 16 pel by 16 line block of the luminance component and the spatially corresponding eight by eight block of each chrominance component. A macroblock may be intra- or inter-coded. An I-picture contains only intra-coded macroblocks. A P- or B-picture may contain intra- or inter-coded macroblocks. The macroblock header contains a macroblock address relative to the previous macroblock in the slice. The absolute address of the first macroblock in a slice is computed by adding its relative address to the address of the last macroblock in the row above the slice's vertical position. The macroblock header also contains information about quantization and may contain a motion compensation vector relative to that of the preceding macroblock. It also codes which of the six possible blocks are actually included in the macroblock (for an intercoded macroblock, blocks which contain no differences need not be transmitted).

block The block is an eight pel by eight line block containing the DCT coefficients. The DC coefficient is differentially coded from that of the preceding block of the same component. If a block is the first block of a slice, the DC predictor is set to zero. The AC coefficients are coded as the number of zero coefficients followed by the next variable-length coded non-zero coefficient.

One additional feature of the MPEG standard is the Video Buffering Verifier (VBV). The VBV is a model that describes the rate at which the encoder may produce bits so that the decoder's buffer never overflows. It consists of a buffer between the output of the encoder and the network. The buffer is of size B where

$$B = \frac{5R}{P}.$$

R is the bitrate of the communications link and P is the picture rate (both constant values for any particular video sequence). The buffer starts out empty, is allowed to fill for an amount of time (specified by the encoder), and then is examined at picture intervals. After instantaneously removing the bits for the oldest picture in the buffer, there must not be more than B bits in the buffer after each picture interval.

Adhering to the VBV model clamps the output rate of the encoder to more-or-less R as a relatively short term average. Unless R is quite large, keeping to this limit will almost certainly entail varying the quality of the picture. Therefore, to use an MPEG encoder with ALF to get a constant picture quality, the ALF application must be able to set the R used by the encoder to limit its rate to some value substantially higher than the average bit rate required by the video. This will allow the encoder to produce large bursts when necessary to accommodate scene changes and other large changes in the video stream.

4.2 Choice of ADU

The slice is the smallest structural unit of MPEG that could be decoded out of order. Macroblocks can depend on earlier macroblocks in the same picture to be decoded, so the macroblock is not the appropriate unit. Pictures are too large to be an ADU. There may be networks where an extremely large ADU is necessary, but it makes no sense to fix the ADU as a picture. Slices, however, do not depend on other slices (or macroblocks) in the same picture. They may depend on earlier (or later) pictures, but that is hard to avoid with any sort of interframe coding for video. Slices are variably sized, from one macroblock to an entire picture, so the ADU size can be varied if necessary. The encoder can size the slices (and hence the ADUs) appropriately depending on the speed and error probability of the intervening link.

Each level of the MPEG hierarchy includes some information in the header for that level. The total length of all this information is about 16 bytes not including the optional quantization tables. Unless the ADU is made to be extremely small, it makes sense to include all this extra information with every ADU. The quantization tables can be transmitted only when they change, in a similar manner to the Huffman and quantization tables for JPEG (see Chapter 3).

4.3 Design of an ALF Application

The transmitting application is not very complex. It must divide the bitstream into slices (having programmed the encoder to produce slices of the desired size), add sequence numbers and timestamps to the ADUs, and send them off to their destination. If the quantization tables ever change, they must be transmitted reliably according to whatever method chosen for them. Finally, if the receiver is performing any sort of retransmission of video data ADUs, the transmitter must be prepared for those requests. This means that ADUs must be buffered until they are too old, where "too old" means that the current clock exceeds the ADU's timestamp plus the transmission, buffering, and decoding delay on the other side. Performing retransmission applies a significant load to the transmitting application.

The MPEG receiving application must perform tasks that are very similar to those of the JPEG application. However, recovering from lost ADUs is much more complex because of intercoded frames. The application must perform the following tasks:

1. Determine whether the ADU is too late. This is the same test as for JPEG. If an ADU arrives too late to be decoded before its video frame is displayed, the ADU is discarded.

2. Install any new quantizer tables that arrive with the ADU in the decoder. The details of this vary with the particular decoder implementation. The tables will be transmitted reliably by one of the methods discussed in Section 2.3.4. As with JPEG, new tables must not be loaded into the decoder until all ADUs that depend on the old tables have been decoded.

3. Deal with lost ADUs. In an intracoded picture, the decoder will not be expecting the gap since the standard specifies that no such gap will exist. Exactly what a given decoder will do in the face of such a gap is therefore uncertain. If there is a context (in other words, the video sequence is being played normally without random access or fast play), then a reasonable behavior is to use the blocks from the preceding frame in the sequence. If the I-picture is there because of a scene change, this approach will produce an ugly glitch. However, there is not really any good way to interpolate a 16 pel high by n pel wide strip from the surrounding pels.

   If the decoder cannot handle the gap in the ADUs, the application must substitute a some kind of placeholder for the missing ADUs. The choices are the same as for the JPEG application.

   If the picture containing the missing ADU is a predictive picture, then the missing slice will be decoded as if the corresponding part of the frame had not changed at all. This behavior is guaranteed by the standard, since a missing ADU is indistinguishable from one that was not transmitted.

   In any case, a lost ADU will affect future (and past) predictive pictures. The error can propagate across the screen at the maximum motion compensation offset per frame (± 96 pels). In the standard code, the damage will not be repaired for certain until the next I-picture arrives (though it is possible for some macroblocks in P- or B-pictures to be intracoded). If there is a reverse channel to the sender, the receiver could request that part of the screen be sent intracoded to fix the problem. If there is enough time, the lost ADU could be retransmitted so that the video picture will not suffer any damage at all rather than being fixed later.

   The difficult part is fixing the correct part of the screen. Since the ADU was lost, some number of predictive (either P- or B-pictures) will have been transmitted. (If an I-picture has been transmitted, there is no longer a need to fix anything.) The receiver can try to calculate the area of the screen that must be repaired either by assuming the worst case, or by storing the motion compensation vectors transmitted with all pictures and macroblocks of the last few frames and figuring out the actual damage. However, the receiver cannot compute the exact damaged area. During the time period between the computation of the affected area and the arrival of the corrections, new intercoded blocks may arrive that depend on the missing ADU, thus expanding the damaged area. A better way is for the receiver to send only a notification of the lost ADU to the transmitter. The transmitter can then calculate the damaged area before assembling and transmitting the correction. Since the transmitter knows what it has transmitted since the lost ADU, it can compute a correction that covers exactly the damaged area.

Given the possible ± 96 pel range of MPEG's motion compensation, a lost ADU may always require that the sender transmit an I-picture. Unless the round trip time to ask for a correction to the damaged area is shorter than one or two frame times (between 15 and 60 milliseconds), the area of possible damage will have expanded to cover the entire screen. The transmitter can limit the rate at which the damage can expand by reducing the maximum motion compensation used by the encoder. If the maximum motion compensation is reduced (for example) to ± 16 pels, the growth of the damaged area is reduced by a factor of 36.

4. Reorder ADUs that need to be reordered. Enough information is included in the MPEG code to allow slices within the same picture to be decoded out of order. However, intercoded pictures depend on other video frames. Therefore, the receiving application must be careful not to send ADUs to the decoder before any frames on which the ADUs depend. The simplest method is to order all the ADUs by frame. The application may send an ADU to the decoder only if the previous frame has been completely sent to the decoder. This limits the

out-of-order processing to be entirely within a frame.

There is a greater potential for out-of-order processing in the MPEG code. In order of increasing complexity, I-pictures can always be decoded in any order; individual intracoded ADUs within a P- or B-picture can be decoded in any order; and individual intercoded ADUs can be decoded if the particular part of their predicting frame has already been decoded. The part of the predicting frame depends on the motion compensation included in the ADU. Furthermore, since the ADU is a slice and the unit of motion compensation is the macroblock, figuring out just which parts of the predicting frame are required to decode a particular slice may become quite complex. Depending on the processing power available, one or more of the preceding constraints can be checked for each newly arrived ADU. To be complete, the application should not only check each new ADU to see if its precedents are satisfied, but also whether it satisfies the precedent(s) of any waiting ADUs.

Finally, the same limitation applies as for a JPEG decoder with several frame stores. An ADU can only be decoded if its frame is currently occupying one of the frame stores of the decoder. Otherwise, there is no place to put its decoded pels, so it must wait.

5. Deal with duplicate ADUs. Unless the network duplicates ADUs with a very high probability, there is no need to worry about the duplicates. If the ADU has already been decoded, decoding it again is an acceptable waste of resources.

The exact details of the application also depend to a great extent on the particular implementation of the decoder. The preceding discussion concerns only the duties of the ALF application as dictated by the code, not the coder. No implementation fast enough to decode (or encode, for that matter) MPEG in real time is yet public knowledge, though C-Cube claims to be working on a chip.

4.4 Multiple Streams

An MPEG decoder's state is the 16 bytes of picture resolution, frame rate, and other parameters mentioned in Section 4.2, the quantization tables used, and several frame stores. Whether it is feasible to use the same decoder for multiple streams will depend mostly on the decoder. As long as it is possible easily to redirect the decoder to the appropriate set of frame stores for the different video streams (perhaps in a similar fashion to the external address register addition to the C-Cube JPEG chip proposed in Chapter 3), and if the video streams use the same resolutions, frame rates and quantization tables, an MPEG decoder can easily be multiplexed among several video streams. Because the ADUs are defined so that they can be decoded out of order, they do not require any low-level state to be retained from one ADU to the next. If the video streams have different resolutions, quantization tables, or other other header information, multiplexing the decoder is further complicated, again depending on the details of the decoder implementation.

Chapter 5

CCITT's $p \times 64$ Video Codec

The CCITT has draft recommendation H.261 [3] for a codec meant for video teleconferencing. It is called $p \times 64$ because it functions at bitrates that are multiples of 64 kilobits.

5.1 Description of Algorithm

The $p \times 64$ code is also an eight by eight pel DCT transform followed by a quantization and variable-length coding of the coefficients. It uses motion compensation and interframe coding. The $p \times 64$ video codec recommendation divides the video stream into the following hierarchical units:

picture Each picture is one video frame. The encoder may save bandwidth by not transmitting some pictures. The picture header contains a *temporal reference* (TR) number which is a five bit frame number (it is incremented by one plus the number of non-transmitted pictures for each transmitted picture), six bits of *ptype*, and room for extra information that is not yet defined. Pictures may be in one of two formats, either CIF (common interchange format) or quarter CIF (QCIF). In each case, the picture consists of one luminance component and two chrominance components. CIF pictures are luminance sampled at 352 rows by 288 lines, and QCIF at 176 by 144. The chrominance components are sampled at half the resolution both vertically and horizontally.

group of blocks Each picture consists of either three (QCIF) or twelve (CIF) *groups of blocks* (GOB). The GOB header contains the address of the GOB in the picture, an integer specifying the quantizer to use for the macroblocks of the GOB, and room for extra information not yet defined.

macroblock Each GOB consists of 33 *macroblocks*, each covering a 16 pel by 16 line area of the picture. Each macroblock header includes the macroblock address, type, quantization, motion vector, and coded block pattern. The macroblock address is relative to the location of the last transmitted macroblock (except for the first macroblock in a GOB, whose address is absolute). The type indicates whether the macroblock is inter- or intracoded, whether a spatial filter should be used on the predictor pels during decoding, and the presence or absence of the optional parts of the macroblock header described next. After the type comes an optional integer (MQUANT) specifying the quantizer to be used for this macroblock and all following macroblocks in the GOB. This overrides the quantizer specified in the GOB header. Following MQUANT is the optional *motion vector data* (MVD). This vector can be relative to the MVD of the preceding macroblock. Next is the optional field *coded block pattern*. The CBP indicates which blocks of the macroblock contain some transmitted data. A block that is intercoded and does not change does not need to be transmitted at all.

block Each macroblock consists of four eight pel by eight line luminance blocks and a single eight by eight chrominance block from each chrominance component. The coefficients of the DCT transform for each block are transmitted run-length encoded in zig-zag order. The end of a block is indicated by a special code that does not otherwise occur in the coefficients. There is no header information in a block.

The bitstream produced by a standard coder contains forward error correction (FEC), though the use of the error correction by the decoder is optional. However, the FEC is over 492 bit blocks of the encoded bit stream—it bears no relation to the other framing in the stream (pictures, GOBs, or MBs). The FEC would be relatively useless for an ALF application using this code, though it may have to be computed and transmitted anyway to conform to the standard. The only reason to transmit or compute the FEC of the $p \times 64$ recommendation is if the decoder used requires it.

The FEC included in the $p \times 64$ standard is the epitome of an anti-ALF philosophy. Since the error correction bears no relationship to the structure of the bitstream, it is impossible to break the bitstream into any sort of unit related to the video frames while retaining the FEC. If the FEC is indeed required to be present in the bitstream produced or consumed by a $p \times 64$ device, the effort to compute it will be entirely wasted. Such an error correcting code is utterly useless for anything but a point-to-point link.

The $p \times 64$ recommendation also places a requirement on the encoder output bitrate that is almost exactly the same as the Video Buffer Verifier of MPEG (see Section 4.1). The only difference is that the buffer size of $p \times 64$ is defined to be

$$B = \frac{4R}{P}.$$

P is fixed at 29.97. As with MPEG, to take advantage of statistical multiplexing and ALF it must be possible to set R to some very large value.

5.2 Choice of ADU

The smallest unit of data that does not have any dependencies on other parts of the bitstream (at least not within a single picture) is the GOB. Macroblocks have addresses relative to preceding macroblocks and the motion compensation vector of a macroblock may depend on a previous macroblock within the same GOB. Blocks do not have any addresses in them at all. Naturally, intercoded macroblocks depend on previous pictures. However, GOBs within the same picture could be decoded out-of-order. There are either three or twelve GOBs in a picture, depending on the format used (CIF or QCIF respectively). GOB headers are transmitted even if no macroblocks within the GOB have any data. The necessary information in picture headers would have to be duplicated in each ADU. That information is the temporal reference number and the format (CIF or QCIF). Additional information might also need to be added by the encoding application, for example sequence numbers and/or timestamps with more than five bits of resolution for the frames. The temporal reference number as specified in the $p \times 64$ recommendation could comprise some part of a larger timestamp. More resolution than a frame time is required in order to be able to timestamp each GOB. Sequence numbers will have to be added to the ADUs so that lost ones can be detected. While the GOB addresses of the recommendation (within a picture) are always transmitted and are sequential, extra bits must be added to differentiate among GOBs from different pictures. Since the number of GOBs in a picture is not a power of two, the best approach may to discard the GOB address (for the purpose of the ALF application) and just add a new sequence number.

5.3 Design of an ALF Application

As with the other codes, the transmitting ALF application is not very complex. It must strip off the FEC, divide the bitstream into GOBs, and add the timestamps and sequence numbers to each ADU. If the receiver will ever ask for ADUs to be retransmitted, the transmitting application must buffer ADUs until they are too old, as described for the other two codes.

The ALF receiving application for a $p \times 64$ video code must perform functions very similar to those of the MPEG application. The $p \times 64$ code is not as complex, but includes both intra- and intercoded frames and motion compensation, leading to similar concerns when handling lost and out-of-order ADUs. The details of the application depend on exactly how the particular decoder used works. The receiving application must perform the following tasks:

1. Determine whether the ADU is too late. This test is the same as for MPEG and JPEG.

2. Deal with lost ADUs. Unlike MPEG, the entire picture is not classified as intracoded or intercoded. Each macroblock can be coded either way. When an ADU is lost, 33 macroblocks are lost each of which may be either inter- or intracoded. Each ADU is either one twelfth or one third of the screen, so any loss will very likely be noticeable. As with the JPEG and MPEG applications, there are a number of possible responses to lost ADUs.

   The application can do as little as possible. In this case, the minimum response is to insert a GOB containing no macroblock data. The decoder will treat this area of the screen as if it had not changed since the previous frame.

   If there is enough time, the application can ask the sender for a retransmission of the lost ADU. If the retransmitted ADU is received in time, all is well. However, the application must be prepared to fall back on some other strategy if the replacement is not received in time.

   Since $p \times 64$ makes it so easy (by inserting a header for an empty GOB in the stream) to use the pels from the preceding frame as a substitute for lost ADUs, it does not make sense to try something like substituting an all-grey ADU for the lost one. A solid rectangle of a single color is almost certain to be more noticeable than the pels from the previous frame.

3. Deal with reordered ADUs. ADUs within the same frame may be decoded in any order, while out-of-order processing across frames depends on the coding

type of the individual macroblocks within the ADU. Each macroblock can have its own motion compensation vector, so figuring out whether all the precedents have been decoded or not is quite complex. Any particular ADU may depend on many ADUs in the previous frame.

There is no restriction inherent in the $p \times 64$ bitstream that makes it impossible or even difficult to decode GOBs within the same picture in an arbitrary order. However, a decoder must not hinder out-of-order decoding. The standard calls for all GOB headers to be transmitted even if they do not contain any macroblocks. Therefore, it is not inconceivable that a decoder could ignore the GOB address in the GOB header and just count on the GOBs arriving in order. Such a decoder would not be very useful for an ALF application. (MPEG does not have this problem because the standard explicitly states that slices do not have to be contiguous. The decoder must look at the slice vertical address to know where in the frame it belongs.)

4. Deal with duplicate ADUs. As with the other codes, as long as the number of duplicates is not extreme, the application should not have to perform any work to eliminate them. As discussed above for out-of-order ADUs, duplicate ADUs may pose a problem if the decoder does not look at the GOB address in the header.

5.4 Multiple Streams

A $p \times 64$ decoder's state is the picture type, the temporal reference number of the last frame received, and two frame stores. As with MPEG, the feasibility of multiplexing a single decoder over several video streams depends on the implementation of the decoder. It must be possible to quickly change the frame store used for each GOB decoded, the picture type (if the video streams are of different types), and the temporal reference number (if the decoder pays attention to it).

Chapter 6

Conclusion

As computers and networks become faster, the digital transmission of video will become a prevalent use of computer networks. The bandwidth available in the immediate future will not be quite enough to send uncompressed digitized video, so the video must be coded in some way that reduces entropy. The amount of redundancy inherent in a video signal strongly suggests some form of compression since even relatively simple compression schemes can produce a significant reduction in the required bandwidth.

Because the redundancy in a video signal changes radically with time, almost all video coding schemes are intrinsically variable rate, and thus stand to profit from the statistical multiplexing gain of a packet network. However, to take advantage of a statistically multiplexed network, a video codec must be able to deal with the other characteristics of a packet-switched network such as jitter and lost packets.

The real-time nature and relatively high bandwidth requirements of video causes a video stream have special requirements as compared to the traffic typically carried on computer networks. The packets must get there in time or they are useless, and the receiver need not process the stream strictly in order. In fact, it will be important that the receiver be able to process the stream out of order, since the decoder will likely be the bottleneck in the communication. If the decoder cannot decode packets out of order, it will fall behind every time a packet is lost. The need for out-of-order processing can arise either because the network reorders packets or because of lost packets.

A new set of network protocol architectural principles called Application Level Framing has been proposed to deal with networks having a high bandwidth-delay product and the idea that the memory bandwidth and/or processing speed of the host will be the communication bottleneck. ALF supports applications with varying service requirements by allowing the application itself to control the response to lost and out-of-order data. The application defines the Application Data Unit (ADU), which is the unit of data that the application can handle out of order. The ADU layer is then responsible only for transporting complete ADUs across the network. The application can be used without change over any network for which an ADU layer can be written.

ALF has many advantages for video. First, it allows efficient implementation of the service requirements of video. The application is not forced to accept levels of service (such as a reliable byte stream) that it does not want. Multiple flows with different priorities or service classes can be created by the application as required. Second, it is efficient in a network-independent way. Previous examples of variable bit rate codecs have been tied to the particular network for which they were designed. They could not otherwise be implemented, since the existing protocols are not efficient enough. Finally, ALF makes it relatively easy to multiplex a decoder over multiple streams. Since each ADU is defined to be as independent as possible, ADUs from different streams can alternately be decoded just as ADUs in the same stream can be decoded out of order. As video becomes more common, users will want desktop video that works just like desktop windows do now. They will want many video streams on the display at the same time, and the ability to move and resize the video windows. If a video codec works with ALF, it can also decode multiple streams.

This thesis has shown how three proposed video coding standards, designed for use over fixed-bandwidth point-to-point circuits, can be made to work over a statistically multiplexed network using the ALF ideas. Sending and receiving applications which perform the necessary extra processing to enable the codes to process ADUs are described. The standards are very similar. Each is implemented using a two-dimensional discrete cosine transform on eight-by-eight blocks of pels, followed by some form of variable length coding of the transform coefficients. The standards differ in that JPEG performs only intraframe coding while the other two also do interframe coding. MPEG and $p \times 64$ differ widely in their complexity.

Even though the coding is inherently variable rate, MPEG and $p \times 64$ are both designed to be CBR codes and as such have a rate limiter on their outputs. The rate limiter must be bypassed or eliminated to achieve the variable bit rate and relatively constant picture quality available with a packet-switched network. Exploring the compatibility of these codes (and coders) with ALF has revealed the following design guidelines:

- The subunits of the bitstream produced by the code should not be addressed by their position in the bitstream. Rather they should contain explicit absolute addresses at some level. This is a requirement of both the code and its implementation. The code should contain the addresses and the coder should use them.

  JPEG, for example, does not contain any addresses in the bitstream, so a decoder (such as the C-Cube chip) that accepts an unembellished JPEG bitstream cannot decode pieces of the bitstream out of order. Luckily, there is nothing else about JPEG that demands in-order decoding, so some additional hardware can be added to the decoder to enable the external ALF application to supply the ADUs with addresses.

  Another example is the requirement of $p \times 64$ that all the group of block headers of a picture be transmitted even if there is no data in a particular group. There is no reason that the GOB header must appear in the bitstream even for a standard decoder; the GOBs contain an absolute address describing their position on the screen. However, since the requirement is in the standard, some implementor may depend on the GOBs arriving in order.

- The code should not include any sort of information in the bitstream that bears no relationship to the semantics of the stream. Of the codes studied in this thesis, the prime example of this onerous behavior is $p \times 64$. The forward error correction that the recommendation calls for is computed and transmitted over 492 bit pieces of the bitstream, which bear absolutely no relation to the rest of the structure of the stream. If the bitstream is divided into units that can stand on their own, the FEC becomes completely useless.

- The code should be as simple as possible given the limits on available bandwidth. Although JPEG requires more bandwidth than the other two, it is much more robust in the face of lost ADUs because the video frames are completely independent of one another. JPEG has no constraints on the order in which ADUs can be decoded. Because of interframe dependencies, the other two codes require either limits on the extent to which ADUs can be decoded out-of-order or complex calculations of the dependencies of ADUs on one another.

  In the immediate future, it seems that interframe coding will be necessary to experiment with video on any large scale because the commonly installed networks are not quite fast enough to support solely intraframe coded video. However, when the bandwidth becomes widely available it makes sense to use simpler video codes.

- The coders and decoders should be designed with multiplexing in mind. The state of the coder should not be hidden away in a black box where it is difficult or impossible to modify quickly. Video stream "context switches" should be as fast as possible. Assuming that the coder implements a well-designed code which has nicely defined units able to be decoded independently, make sure that the decoder can actually decode them independently.

This design issue depends both on the code and the implementation, so it is difficult to compare the three codes since there is only one implementation: the C-Cube chip. For video streams which operate in the same modes, have the same resolutions, and use the same Huffman and quantization tables, the chip can decode multiple streams with the addition of some external hardware. For the other two codes, the ability to decode multiple streams is less clear. It is possible to build a codec that could easily handle multiple streams—there is no great amount of state that would have to be swapped—but there are not any implementations to examine.

The ideal video code for use with an ALF protocol would code each picture into relatively small units. Each ADU would be independent of all the others in at least the same picture. If the bandwidth is available, each ADU would be entirely independent of all other ADUs of the video stream (in other words, intraframe coded). If network limitations dictate that intercoding be used, then it should be coded as a straight forward difference from the preceding frame with some relatively limited motion compensation. The processing necessary to compensate for lost ADUs should be as simple as possible while allowing the picture quality to degrade as little as possible.

Of the codes studied in this thesis, the one that best meets these criteria is JPEG. If there is enough network bandwidth available to carry it, JPEG is the simplest and easiest to use. None of its ADUs depend on one another to be decoded and recovering from lost ADUs can be accomplished relatively simply by replaying the ADU from the previous frame.

If an interframe code must be used because of bandwidth limitations, the best of the other two codes in $p \times 64$. MPEG's complexity does not provide any compelling advantage for real time video transmission. If storage space for the video is at a premium and the encoding process can take a very long time, then the addition flexibility of MPEG's many picture types and large motion compensation might provide a small advantage. However, for use over a network that might lose or delay packets, $p \times 64$ is much simpler.

Unfortunately, none of the codes studied come as close to the ideal as is possible. MPEG and $p \times 64$ require complex strategies to cope with lost ADUs, and careful consideration of which ADUs may be decoded out-of-order. JPEG is better because its ADUs are entirely independent, but substituting from the previous frame for lost ADUs can unacceptably degrade image quality during a scene change.

Any packet-switched network will experience packet loss to some degree. Network congestion may cause packets to be dropped and transmission bit errors can cause packets to be misdelivered or to fail error checks. Therefore it is very important that a video code for use over a packet network be able to cope with packet losses without severely degrading the picture quality and without requiring excessive computation. A family of video codes that provides very good error recovery is the family of so-called layered codes. A layered code is one that divides the video data into layers of varying importance. An example of a layered code is a subband code. A subband code filters the video into different frequency bands and codes each band separately. The baseband is very important to the picture quality while the other bands are much less important. If the network provides different priorities, each band can be transmitted at a different priority. Then ADUs from the baseband will not be lost due to network congestion. The ALF architecture provides an ideal framework for such a code because it explicitly allows the application to specify the service class desired for each flow.

Of the three codes studied in this thesis, JPEG is the only one that could produce a layered bitstream. Either its hierarchical or progressive modes could be used to divide up the data into layers. However, JPEG does not suffice if an interframe code is needed. Also, using a code that does not require a DCT can be helpful for handling packet losses. If the data that are lost are in units of eight-by-eight blocks, it is not realistically possible to fix the error through interpolation from the surrounding pels. Assuming that retransmission is not possible, the best correction is to repeat the pels from the previous frame. However, this correction will be very noticeable if it occurs during a scene change. It is more desirable to interpolate missing pels from the surrounding area of the same frame.

The best code to use would be a subband code similar to the one described in [6]. Such a code lends itself to a network that can give different priorities to different flows, can easily be divided up into ADUs, and recovers from lost ADUs gracefully. Subband coding provides other advantages as well. The resolution of the displayed picture can be varied depending on the number of bands uses. If the user makes the displayed video window be smaller, the receiver can ask the sender to send only the lower bands, to reduce the load on the network (and on the receiver) by not transmitting what will not be needed. If the network becomes congested, the sender or network can also reduce the number of bands transmitted.

If video codes are designed to follow the above guidelines, they can make use of an ALF protocol and thus will be usable without modification over a wide range of networks. ALF makes it possible to efficiently satisfy the service requirements of video.

Bibliography

[1] R. C. Brainard and J. H. Othmer. Television compression algorithms and transmission on packet networks. In T. Russel Hsing, editor, *Visual Communications and Image Processing '88*, pages 973–978, 1988. Proc. SPIE 1001.

[2] CL550 JPEG image compression processor, November 1990. C-Cube Microsystems Part Number 90-1150-202.

[3] Video codec for audiovisual services at p x 64 kbit/s. International Telegraph and Telephone Consultative Committee, August 1990. Study Group XV, Report R37, Recommendation H.261.

[4] H. Jonathan Chao and Cesar A. Johnston. Asynchronous transfer mode packet video transmission system. *Optical Engineering*, 28(7):781–788, July 1989.

[5] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings ACM SIGCOMM*, pages 200–208, September 1990. Philadelphia.

[6] John C. Darragh and Richard L. Baker. Fixed distortion subband coding of images for packet-switched networks. *IEEE Journal on Selected Areas in Communications*, 7(5):789–800, June 1989.

[7] James R. Davin. ALF spec working paper. Work in progress.

[8] Luís de Sá and Vitor Silva. Implementing a 64kbit/s video codec on DSP hardware. In *Applications of Digital Image Processing XII*, pages 90–95, 1989. Proc. SPIE 1153.

[9] Herbert Hölzlwimmer, Walter Tengler, and Achim v. Brandt. A new hybrid coding technique for videoconference applications at 2 mbit/s. In Murat Kunt and Thomas S. Huang, editors, *Image Coding*, pages 250–259, 1985. Proc. SPIE 594.

[10] K. Iinuma, T. Koga, K. Niwa, and Y. Iijima. A motion-compensated interframe codec. In Murat Kunt and Thomas S. Huang, editors, *Image Coding*, pages 194–201, 1985. Proc. SPIE 594.

[11] Information Processing Systems — Open Systems Interconnection: Basic Reference Model, 1984. Interational Standard 7498.

[12] Revision 8 of the JPEG technical specification. Joint Photographic Experts Group, August 1990. ISO/IEC JTC1/SC2/WG8, CCITT SGVIII.

[13] S. H. Lee and L. T. Wu. Variable rate video transport in broadband packet networks. In T. Russel Hsing, editor, *Visual Communications and Image Processing '88*, pages 954–964, 1988. Proc. SPIE 1001.

[14] Robert J. Moorhead, Joong S. Ma, and Cesar A. Gonzales. Realtime video transmission over a fast packet-switched network. In Ying wei Lin and Ram Srinivasan, editors, *Digital Image Processing Applications '89*, pages 118–123, 1989. Proc. SPIE 1075.

[15] Coding of moving pictures and associated audio (MPEG). International Organization for Standardization/International Electrotechnical Institute, September 1990. ISO/IEC JTC1/SC2/WG11.

[16] Hans Georg Musmann, Peter Pirsch, and Hans-Joachim Grallert. Advances in picture coding. *Proceedings of the IEEE*, 73(4):523–548, April 1985.

[17] Arun N. Netravali and Barry G. Haskell. *Digital Pictures: Representation and Compression.* Plenum Press, 1988.

[18] Philips International, Inc., editor. *Compact Disc-Interactive: A Designer's Overview.* McGraw-Hill, 1988.

[19] G. David Ripley. DVI—a digital multimedia technology. *Communications of the ACM*, 32(7):811–822, July 1989.

[20] Eve M. Schooler and Stephen L. Casner. A packet-switched multimedia conferencing system. *ACM SIGOIS Bulletin*, 10(1):12–22, January 1989.

[21] Willem Verbiest and Luc Pinnoo. A variable bit rate video codec for asynchronous transfer mode networks. *IEEE Journal on Selected Areas in Communications*, 7(5):761–770, June 1989.

[22] Willem Verbiest, Luc Pinnoo, and Bart Voeten. The impact of the ATM concept on video coding. *IEEE Journal on Selected Areas in Communications*, 6(9):1623–1632, December 1988.

[23] Masahiro Wada. Selective recovery of video packet loss using error concealment. *IEEE Journal on Selected Areas in Communications*, 7(5):807–814, June 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1992	3. REPORT TYPE AND DATES COVERED
4. TITLE AND SUBTITLE Video Coding and the Application Level Framing Protocol Architecture		5. FUNDING NUMBERS
6. AUTHOR(S) Heybey, A. T.		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Massachusetts Institute of Technology Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139		8. PERFORMING ORGANIZATION REPORT NUMBER MIT/LCS/TR-542
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA 1400 Wilson Blvd. Arlington, VA 22217		10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA Grant NAG 2-582
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT		12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

As networks and computers become faster, real time video transmission is expected to become common. Variable bit rate video coders will be used in order to take advantage of the statistical multiplexing gain and bandwidth efficiency of packet switched networks. Video streams have different service requirements from the traffic usually carried on computer networks. A new protocol architecture called Application Level Framing (ALF) has been proposed to allow efficient implementation of communications with diverse service requirements. ALF allows the application to control the way in which network errors are handled. This thesis studies the compatibility of three proposed video coding standards with an ALF protocol architecture. Each of the standards is found to be usable with varying degrees of effort. A set of design principles for video codes intended for use over an ALF protocol architecture is presented.

14. SUBJECT TERMS			15. NUMBER OF PAGES 67
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT