RC 20532 (08/28/96)
Computer Science

IBM Research Report

Robust H.263 Video Coding for Transmission over the Internet

M. Willebeek-LeMair, Z.-Y. Shae

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York
mwlm, zshae @watson.ibm.com

Y.-C. Chang

EECS Department
University of California, Berkeley
Berkeley, CA 94720
yuanchi@eecs.berkeley.edu

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Robust H.263 Video Coding for Transmission over the Internet

Marc H. Willebeek-LeMair†, *Zon-Yin Shae*†, *and Yuan-Chi Chang*‡

†IBM Research
Thomas J. Watson Research Center
Yorktown Heights, NY 10598
mwlm, zshae@watson.ibm.com

‡EECS Dept.
University of California, Berkeley
Berkeley, CA 94720
yuanchi@eecs.berkeley.edu

September 9, 1996

Abstract

The widely popular World Wide Web along with advances in desktop computers has brought the world into a new age of computing and communications. Low bit-rate video applications across the Internet are quickly emerging. The ITU-T H.263 standard was designed for low bit-rate video conferencing across phone lines and is an ideal candidate to be extended for Internet video applications. A primary issue in transmitting compressed video streams over the Internet is to achieve fast recovery from packet losses while maintaining a smooth bandwidth profile. This paper focuses on the error robustness issue of compressed H.263 video streams. The traditional approach of inserting intracoded frames increases the error resilience at the expense of bursty output traffic, lower picture quality, and uneven frame dropping. By extending the macroblock force update feature of the H.263 standard, we developed a scheme that complies with the standard and increases the robustness of the video stream. This macroblock updating scheme analyzes the temporal dependencies of macroblocks in successive frames and selectively updates the macroblocks which have the most impact on later frames. The performance evaluation of the proposed technique demonstrates that it achieves a good balance between error recovery speeds and video quality.

1 Introduction

As the widely popular World Wide Web evolves and becomes a part of our daily life, and powerful multimedia desktops replace the images of dumb, typewriter-like terminals, the two forces have been merging to create a new era of communications and computing. At the center of the stage is the ability to access and communicate in multimedia, using audio, video, images and data. Today's personal computers have reached the point where computationally intensive signal processing functions can be performed in software rather than expensive hardware. Despite many technological advances in communications, an increase in residential communication bandwidth has been slow to evolve. Many people still rely on phone lines to access the Internet. The available bandwidth of a phone line is more than a thousand times lower than the bandwidth of an uncompressed NTSC video stream. The need for efficient low bit-rate video coding thus arises.

The ITU-T H.263 video compression standard [1][2] provides a solution to the need for low bit-rate video compression. It exploits the temporal and spatial redundancies in a video sequence to greatly reduce the required bandwidth. For example, two consecutive frames in a video sequence often differ slightly. Compression techniques can thus take advantage of this redundancy by only encoding the differences between the two frames. The differential coding, however, creates dependencies between successive frames, which may cause error propagation when video data is lost in the middle of the transmission. To prevent serious quality degradation, it is important to increase the error resilience of the compressed stream.

In this paper, we focus on a class of low bit-rate Internet video applications like video playback and video multicast. These applications require fast recovery from packet losses and do not depend on special network transmission or error control protocols. We show that by intelligently refreshing parts of each video frame, the time to recover from errors can be significantly reduced while the output stream maintains a smooth bandwidth profile.

The Intelligent Macroblock Update (IMU) compression technique defined in this paper is fully H.263 standard compliant and has several key features:

1. Fast recovery from packet-losses. The IMU compression technique reduces the propagation of errors caused by lost or corrupted data, allowing the image to recover quickly.

2. Short join latency. For multicast applications, when stations join the multicast session mid-stream, the IMU technique enables the video image to converge quickly.

3. Fast-forward and rewind functions. When seeking through a video sequence the

IMU technique enables the video playback to resume quickly from any arbitrary point in the sequence.

As will become clear shortly, the mean packet loss interval and the expected session join latency are used to determine a coding parameter in the IMU scheme. These interval values are assumed to be known a priori. We further assume video data is carried by UDP packets, which have no guarantee to arrive correctly. Error-free communications such as TCP may be effective for point-to-point non-real-time applications. However, for video streaming and multicast situations, a non-guaranteed transport layer without retransmission delays is preferred. The simple UDP transmission approach also reduces the complexity of the video server design and makes it more efficient.

There is a significant amount of prior research on error resilient video coding. The proposed schemes vary from signal processing approaches such as the enhancement of the codec architecture [3]-[7], to communication approaches such as bit stream packetization and error control coding [8]-[10]. Combinations of signal processing and communication solutions have also been proposed [11]-[15]. The ultimate solution, however, often lies in the understanding of the nature of the video application, its performance requirements, its target transmission media, and the system limitations. For example, forward error correction (FEC) is more effective at handling random bit errors and less effective at handling packet losses. Non-real-time applications prohibit the use of closed-loop encoding control, like adaptive resolution and quality adjustments, which must adapt to channel conditions in real time. In multicast scenarios, closed-loop error control may not be feasible due to scalability issues. Special data packetization techniques based on the compressed stream structure may be a favorable choice for real-time unicast applications but can create unacceptable processing overheads on large scale video servers.

The proposed IMU solution for our target applications is constrained by several design factors. The multicast support rules out the possibility of employing closed-loop error control. The congestion-oriented loss mechanism of the Internet makes FEC less appealing. The video server overhead limitations prohibited us from deploying retransmission transport schemes or from designing special packetization schemes. Finally, the requirement of H.263 standard compliance forces our scheme to work within the limitations of the standard algorithm.

The IMU compression technique proposed in this paper employs the macroblock force update feature of the H.263 standard to achieve the aforementioned performance objectives. The scheme is devised to intelligently choose the important macroblocks, which have more impact on successive frames than others, and update them more frequently to ensure their correctness. An overview of the H.263 encoder architecture is given in Section 2 along with a discussion of different updating strategies. Realizing the insufficiency of simple static macroblock updating schemes, we propose a more elaborate algorithm for non-real-time encoding to determine when and where to insert updated blocks. A detailed description of the IMU scheme is given in Section 3. We present the performance evaluations of the proposed algorithm in Section 4. Section 5 discusses a heuristic solution developed for real-time encoding. Section 6 concludes the paper.

2 The H.263 Encoder Architecture and Its Error Resilience

The ITU-T H.263 recommendation is a part of a suite of standards designed for low bit-rate videophone systems over PSTN. Derived from the earlier H.261 standard (designed for ISDN at 64kbps to 2 Mbps), H.263 is designed to fill the coding rate gap below 64 kbps. H.263 was targeted for video conferencing applications, where the video content is mainly a "talking head" scene and the motion is generally slight. From this type of content a significant amount of compression gain can be achieved by eliminating spatial and temporal redundancies. Like MPEG, H.263 also uses motion compensation and the Discrete Cosine Transform (DCT) approach to compression. The encoding algorithm can be decomposed into four steps: motion estimation, transform coding (DCT), linear quantization, and run length coding.

A block diagram of the H.263 encoder is shown in Fig. 1. The encoder first partitions a video image into a set of macroblocks of 16x16 pixels. For each macroblock, the coding control function searches for the best match between the macroblock and macroblocks in a previous frame. If a close enough match is found, the macroblock is encoded using a motion vector and the difference between the macroblock and the closest matching macroblock in a previous frame. This is referred to as an inter-coded macroblock since it has dependencies on another macroblock. A "close enough" match is evaluated in terms of the number of bits needed to encode the difference. If the difference is too great then it is deemed more efficient to encode the macroblock independently without reference to previous data. This is referred to as an intra-coded macroblock. An inter-coded block is often abbreviated as a P-block and an intra-coded block is abbreviated as an I-block.

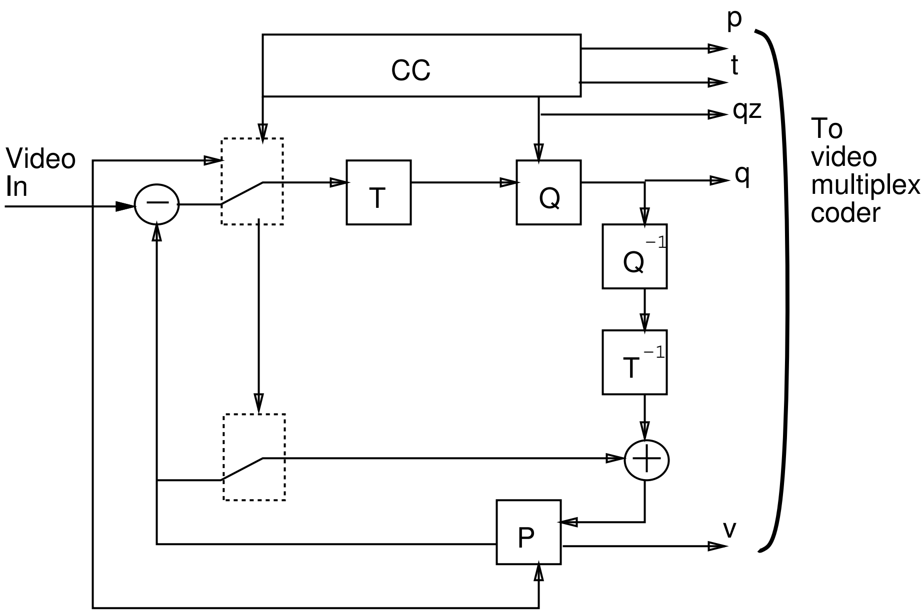
The resulting H.263 compressed stream consists of mostly P-blocks with phases of I-block insertions caused by scene changes or severe motion. To prevent error accumulation, the standard also requires that each macroblock be encoded as an I-block at least every 132 frames. Hence, the H.263 standard defines how I-blocks and P-blocks are encoded but allows considerable flexibility in selecting when to encode a block as either an I or P block. We exploit this flexibility to improve the robustness of a standard compliant stream by carefully choosing when and where to insert I-blocks during the encoding process.

I-block encoding exploits only the spatial redundancy within the block while P-block encoding exploits both the temporal and spatial redundancies of the video. Even though it generally achieves more compression gain, the encoding dependencies of P-blocks reduce their resilience to errors. If the region referenced by a P-block is corrupted at the decoder, the decoding of the P-block will generate incorrect pixel values. If a whole or part of the P-block is again referenced by other P-blocks, the erroneous pixel values will propagate from one frame to another. This is known as the error propagation problem of motion compensated video compression. The propagation stops when all corrupted regions are updated by I-blocks, which do not depend on previous frames.

On the Internet, where loss is primarily attributed to network congestion, the loss of an IP packet which contains video data can translate to a loss of several frames in a low bit-rate video application. For example, the average compressed video frame size is 165 bytes for a coding rate of 20 kbps and a frame rate of 15 fps. Hence, a 500-byte long IP packet contains roughly three frames of compressed video data. If the packet is lost, these frames cannot be recovered and errors start propagating.

I-blocks increase the robustness of a video stream, but also require more bits to encode. The most robust video encoding uses I-blocks only, or equivalently encodes all video frames as intra-coded frames (I-frames). This is the same as encoding the video sequence as a series of independent images. Missing one image does not affect the decoding of other images. This scheme, however, drastically reduces the compression gain and with the low bit rate constraint, the frame rate is forced to drop. The picture quality may also degrade because higher quantization levels are used to reduce the number of bits needed to encode an I-frame.

Increased video encoding robustness can also be achieved by periodically refreshing the image by inserting I-frames. Because an I-frame does not have a dependence on previous frames, it can effectively terminate any error propagation. In the MPEG standard, every Group-of-Picture (GOP) sequence has an I-frame. The size of a



T Transform
Q Quantizer
P Picture Memory with motion compensated variable delay

CC Coding control
p Flag for INTRA/INTER
t Flag for transmitted or not
qz Quantizer indication
q Quantizing index for transform coefficients
v Motion vector

Figure 1: A schematic diagram of the H.263 encoder.

GOP is not defined but the most commonly used values range between 12 and 15. Therefore, error propagation is terminated in at most 15 frames. The purpose of having the GOP in MPEG is mainly for video browsing but it also increases the robustness of the MPEG stream. For low bit-rate applications, however, the scheme of I-frame insertions is less favorable because it generally requires many more bits to encode an I-frame than a P-frame. Since an I-frame insertion tends to use up the bit rate budget of multiple frames it can cause the encoder to skip several consecutive frames, which creates motion jitter as the video plays back.

The third way to improve robustness in the compressed video stream is to insert I-blocks incrementally to refresh the image. This feature in H.263 is used in place of I-frame insertions. In H.263, I-blocks are used to update portions of a frame incrementally to achieve a complete update. This helps maintain smoother processing and bandwidth requirement between frames. To recover from the accumulation of DCT mismatch errors, the standard requires I-block insertions at least once every 132 coded frames and this maximum insertion interval is defined as the force update period. By shortening the force update period, the error recovery can be improved. This scheme, however, does not produce fixed synchronization points. I-block insertions are scattered across different frames and one cannot get the video totally reconstructed in one frame.

The tradeoff among the above three solutions for a fixed target bandwidth, is between video quality, smooth bandwidth utilization, and error recovery speed. The first method generates the poorest quality video (low frame rate and coarser quantization) but it always recovers from errors in one frame. The second method generates bursty traffic and it recovers from losses in one I-frame refresh period. The video may be degraded when an I-frame is inserted if the bandwidth constraints force a coarser quantization for the I-frame encoding. The third method generates a smooth bandwidth profile but may take long to recover from packet losses.

For most Internet users to benefit from a network video application, we believe it is necessary for the application to generate smooth traffic to be carried by phone lines, which are constant bit rate. Therefore, we developed our solution in the direction of the third method, which generates the least bursty traffic.

Our scheme exploits the flexibility within the H.263 standard to select when and where to insert I-blocks within the compressed stream. To make the incremental macroblock updating more efficient and effective, certain information such as motion vector dependencies in a video sequence can be exploited. The next section introduces our scheme, intelligent macroblock updating (IMU).

3 IMU for Non-Real-Time Encoding

Our intelligent macroblock updating (IMU) scheme for inserting I-blocks, not only maintains the low bit rate requirement of the application, but also increases its robustness to packet loss. More importantly, the compressed video stream is compliant with the H.263 standard. The main idea is to select macroblocks to be updated intelligently based on their impact on future frames. I-blocks are updated incrementally as opposed to the I-frame updating at fixed intervals. Some locations in a frame may be updated more frequently than others and some frames may contain more I-blocks than other frames.

The delay tolerance of an application has significant impact on the choice of updating schemes. The main difference between a real-time scheme and a non-real-time one is the ability for the encoder to look "ahead" given the delay constraints. Since macroblock dependence in a sequence is hidden in future frames, the farther ahead the encoder can access, the better the I-block insertion decisions it can make. The cost of looking ahead comes in terms of the increase of picture buffer size and its associated encoding delay. In the following, we first discuss the solution for non-real-time applications.

A primary objective of the scheme is fast convergence following the receipt of corrupted data or the occurrence of lost data. Before addressing the speed of convergence, it is important to question if it will converge at all. Due to the error propagating nature of motion compensation, incremental updating may never reconstruct an error-free frame. This happens when the motion vectors always point to corrupted areas. We are able to prove, under the assumption that motion vectors are uniformly distributed, that a corrupted video stream with incremental updating will eventually recover from errors, provided that the sequence is long enough. This result applies to all kinds of updating schemes, regardless of their updating periods or updating orders. The proof is given in Appendix A. For a video sequence of finite length, the distribution of motion vectors may not truly resemble a uniform distribution. However, our simulation results based on several test sequences do exhibit this convergence property.

It is important to note that even though the incremental updating converges no matter what strategy is chosen, the convergence speed varies. We conducted simulations to evaluate several simple updating techniques which will be discussed in the next section.

For non-real-time applications, knowledge about the interdependence among mac-

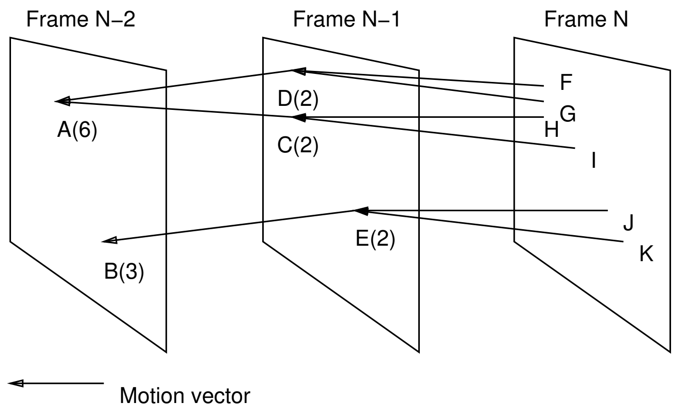


Figure 2: An illustration of constructing a dependence graph. The number in the braces is the dependence count of a pixel.

roblocks in a sequence can be obtained by analyzing motion vectors which reference the previous occurrences of macroblocks. The previous frame, again, has dependence on the frame before that. The dependence connections form a graph in the time domain. It is thus possible to assign a measure of importance to a pixel or macroblock by counting the number of pixels or macroblocks which depend on it. This operation is anti-causal, i.e., traversing backward in time. The importance of finding those critical macroblocks lies in the fact that errors occurring in those blocks may cause the greatest picture quality degradation due to error propagation. It is thus important to ensure their correctness.

An example of how to construct a dependence graph and calculate dependence counts is presented in Fig. 2. Starting with the last frame N, in a sequence, one can determine the dependence on pixels in the previous frame. In Fig. 2, pixels F and G in frame N reference pixel D in frame $N-1$, since both motion vectors point to D. Similarly, pixels H and I reference pixel C. We thus define the dependence count of pixel D to be two, which means two pixels in subsequent frames reference D. Pixel A in frame $N-2$ has a dependence count of six because pixels C, D, F, G, H, and I depend on it. If A is corrupted, the error spreads from one pixel to four pixels in just two frames. The topology of the graph, created by linking the motion vectors, is a directed tree structure. We define this graph as a dependence tree.

The intelligent macroblock selection procedure sets a target number for the I-blocks to be inserted. This target value depends on the packet loss interval and session join latency discussed earlier and has an effect on the frame rate in a fixed bit rate environment. In general, the more I-blocks inserted, the more robust the video and the lower the frame rate generated. To begin, an arbitrary dependence threshold is selected. This dependence threshold represents the maximum number of pixels that may depend on a block. Should a block's dependence count exceed the threshold, it is converted to an I-block. The effect of converting a block to an I-block is a segmentation of the dependence graph. After a complete pass of the video sequence, the total number of I-blocks may be above or below the target. The algorithm then adjusts the threshold accordingly and iterates. The iteration continues until the target range is reached or the maximum number of iterations allowed is exceeded.

A two-pass encoding process is employed. In the first pass, the dependence information is acquired and the positions of I-blocks are determined. The second pass executes the compression using the force updates determined through analysis of the first pass. The apparatus used to implement the two-pass encoding is illustrated in Fig. 3. In the original H.263 standard, the encoder only includes Stage I in this diagram. Due to our enhancement, Stage II and III are added, including the picture buffer and the IMU module. As video comes in, it gets buffered and is encoded as usual in Stage I. However, the coded data is not sent out. Instead, the motion vector information is extracted and fed into the updating analysis component within the IMU. The IMU gives instructions on I-block insertion locations to the coding control block of Stage III, which then encodes based on these decisions.

A detailed description of the IMU algorithm is illustrated as a flow diagram in Fig. 4. In the first pass, the encoder sets a force update period (FUP), which specifies the maximum number of frames between two consecutive updates of a given block. The FUP is set not to exceed 132 in the H.263 standard. In our scheme, it depends on the packet loss interval and the session join latency of video multicast. The first constraint requires that the decoded stream fully recover from a packet loss before the next loss hits the data. Assume that each packet contains N frames of compressed video. Given a packet loss frequency of one in every P packets, the recovery has to complete in $N * (P - 1)$ frames. Based on our experimental results, it takes approximately eight times the FUP to reconstruct a correct frame. One can, thus, calculate the maximum FUP to be around $N(P - 1)/8$ frames. For a one-percent packet loss rate ($P = 100$) and $N = 3$, the maximum FUP is 37 frames. The other factor on the choice of the FUP is the join latency of video multicast. Let us assume, a join latency of ten seconds. This is similar to the situation where the decoded sequence must recover from a packet loss in ten seconds. Assume the display rate is 15 fps. The maximum FUP is 18 frames, which is smaller than the average interval

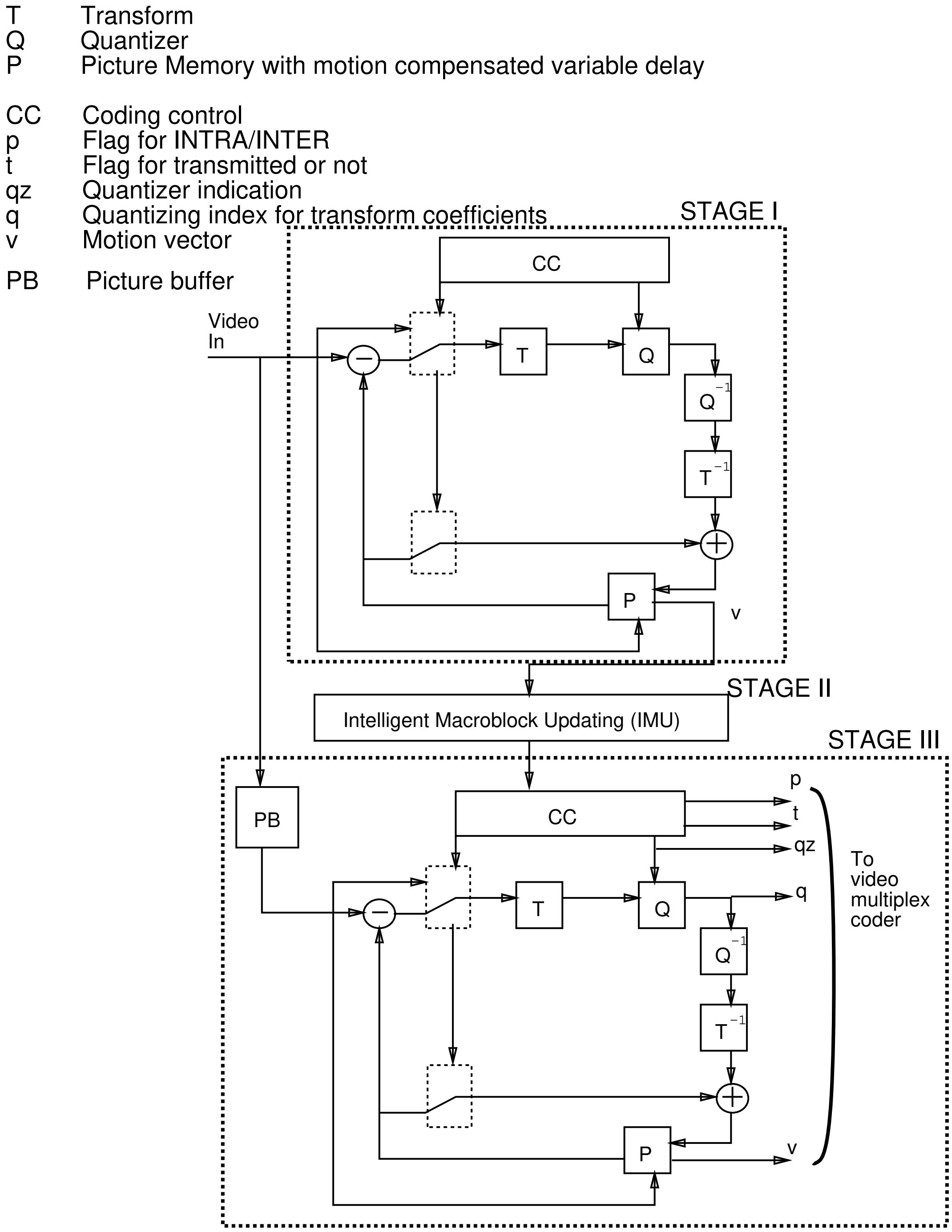


Figure 3: The schematic diagram of the non-real-time encoder with intelligent macroblock update.

(P) between lost packets for the assumed packet loss frequency. The FUP set in the first pass is the minimum of the two values. In this example, it is set to 18 frames.

In the current implementation, in addition to the motion vectors, two more parameters are extracted in the first pass. These two parameters are the total number of inserted force update blocks, α, and the maximum number of inserted blocks in a single frame, β. In the second pass compression, α is used as the target number of I-block insertions. The operation can be interpreted as rearranging their temporal and spatial locations to maximize the robustness to errors without increasing the bandwidth. The reason the target insertion number is selected in this way is twofold. The first is that a video sequence may be inherently robust, provided that it has frequent scene changes and a lot of naturally generated I-blocks in the compressed stream. In this case, additional I-block insertions required may be few if any at all. Other sequences with less scene changes may be less robust and will require more I-block insertions. Therefore, α is not fixed but is content dependent and must be determined at run time.

The second reason for extracting the two parameters is more related to the interaction between rate control functions and I-block insertion decisions. One important assumption for the two-pass compression is that the second pass will encode the exact same set of frames as the first pass did. Without the intervention of rate control, the assumption is correct. However, if the updating analysis inserted too many I-blocks in a single frame, the rate control module might skip frames which were coded in the first pass compression. The dependence graph built on the set of frames from the first pass is then incorrect and the second pass compression will insert I-blocks at incorrect locations and diminish the coding efficiency. It is thus important to avoid this situation by limiting the maximum number of I-block insertions to be the same as in the first pass.

The computational complexity of the IMU algorithm is a function of the size of each frame and the length of the sequence. Assume the size of the frame is P by Q pixels and the sequence has N frames. The calculation and update of dependence counts for all the pixels in a frame requires $2 * P * Q$ additions. This includes $P * Q$ additions to update the dependence counts of pixels and another $P * Q$ additions to sum up the dependence counts of macroblocks. All but the last frame in a sequence are updated. Therefore, an iteration requires approximately $2 * (N - 1) * P * Q$ additions. The computational complexity of the algorithm is thus $O(NPQ)$. The memory usage of the algorithm also grows linearly as the sequence length increases.

In certain cases, it might be infeasible to process the whole sequence before starting to encode it. With a little modification, the same algorithm can operate on a window-by-window basis. The encoder performs the two pass coding on all frames in a window by assuming there is no dependence on the last frame. It then moves to the next non-overlapped window and encodes in the same way. This technique cannot be applied to delay-critical applications when frames cannot be pre-buffered. A scheme to deal with real-time applications is discussed in Section 5.

4 Performance Evaluation and Discussions

The experimental performance results are important in evaluating the relative merits of the IMU scheme in comparison to other techniques. First, a simulation experiment is used to explore the effect of various static update patterns on the convergence speed, given no knowledge of the video sequence. The second part of the evaluation is to compare the IMU scheme against the fastest static update pattern. Two major evaluation criteria are the convergence speed, and the number of frames encoded. Typically, a fast converging scheme, which inserts more I-blocks, causes more frames to be skipped at the encoder and thus generates a lower frame rate video. However, this relation is non-linear and the purpose of this section is to demonstrate that the IMU scheme only skips a few more frames in exchange for a significant gain in the convergence speed.

The first set of simulations were performed to test different static update patterns to find which converges the fastest. The five patterns tested, illustrated in Fig. 5, are: (a) sequential updating in the Y direction; (b) sequential updating in the X direction; (c) random sequence of block updates, where all blocks are updated before a block is updated for the second time; (d) updating from four corners to the center; and (e) updating from the center to the rest of the frame. We assume a QCIF(144x176) frame-size sequence is updated starting with a totally corrupted frame. A QCIF image contains 99 macroblocks. Five macroblocks are updated per frame, so it takes about 20 frames to refresh the entire image. We further assume that no losses or errors occur in the received data after the first corrupted frame. All motion vectors are uniformly distributed in the motion search area.

The simulation results are presented in Fig. 5. Since the convergence speeds of the five patterns are fairly close, the curves are plotted in four separate graphs. Each graph plots the fastest updating pattern and one other pattern. For each test pattern, the curve shows the median values of 150 randomly generated test cases. The frame number is plotted along the horizontal axis, while the percentage of correct pixels in

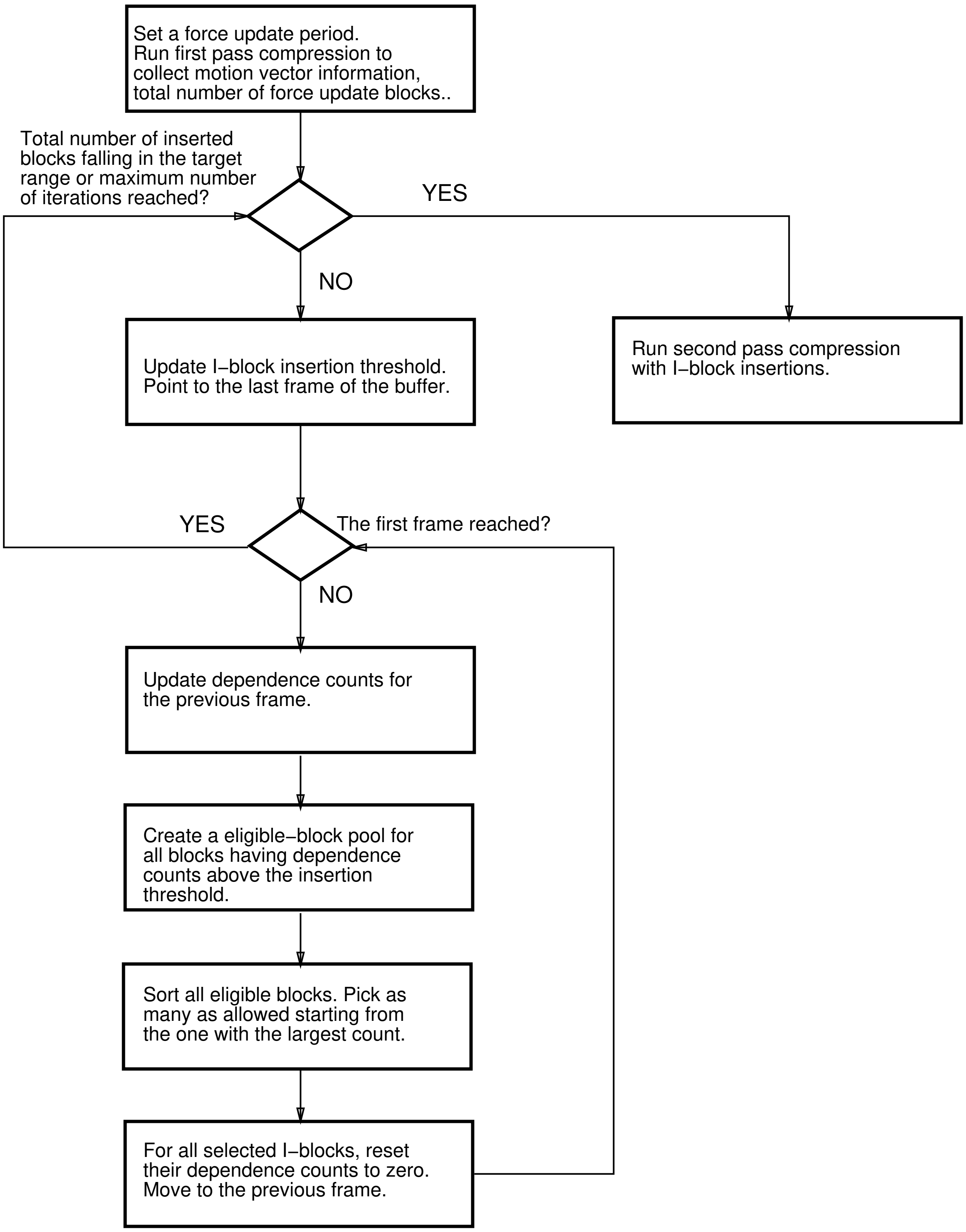


Figure 4: The flow diagram of the intelligent macroblock updating algorithm.

a frame is plotted along the vertical axis. Among the five patterns, both sequential updating schemes (a) and (b) generate the fastest convergence speed and the center updating (e) is the slowest. Intuitively, whichever pattern manages to minimize the boundary between updated and corrupted regions, converges the fastest. The center updating has the longest such boundary, increasing the probability that an error in a corrupted block propagate to a non-corrupted region.

The second set of simulations are conducted to compare the IMU scheme with the static sequential update strategy. Three H.263 test sequences, Suzie, Miss America, and Carphone, are used. Each test sequence has 500 frames and contains mild motion. The bit rate is set to 20 kbps with a target frame rate of 15 fps. The rate control function is in effect. To make a fair comparison, the forced update period is set to 20 frames for both schemes. In all the tests, the second encoded frame is dropped and is replaced by a gray image, in which each pixel has a value of 128 for the Y, U and V components. This simulates the situation when a user joins a multicast session and starts reconstructing the sequence from a blank screen. Fig. 6 shows three sets of image frames from our simulation. The images on the left side are the decoded images of the IMU scheme, while the ones on the right side are the decoded images of the static update strategy. Fig. 6(a), (b) and (c) show the images of the 50th, 100th and 200th frames, respectively. The most noticeable artifacts are the scattered gray areas in the background, which should be dark.

The visual artifacts are reflected in the numerical measures as well. Fig. 7 shows the PSNR curves of the IMU, static, and un-corrupted sequences. The un-corrupted one is used as a reference. The horizontal axis is the number of frames after the corruption. Both U and V components converge fairly quickly due to the fact that the size of the U and V planes is a quarter of the size of the Y plane. As shown in Fig. 7, the convergence speed of the static update strategy is much slower than that of the IMU scheme. Other tested sequences produced a similar result.

Next, we plotted a graph of the convergence speed versus the number of encoded frames for five error resilient schemes. Ranking the most robust to the least robust ones, they are (a) all the frames are I-frames; (b) one I-frame in every 20 frames; (c) the IMU scheme with FUP equal to 20; (d) the static updating strategy with FUP equal to 20; and (e) the original, unprotected case with FUP equal to 132. We define the convergence time of a sequence to be the time that its PSNR is within the $3dB$ range of the PSNR of its un-corrupted counterpart.

For scheme (a), any errors can be recovered in one frame. But out of 500 frames in the Miss America sequence, the encoder only generates 87. This is due to the fact

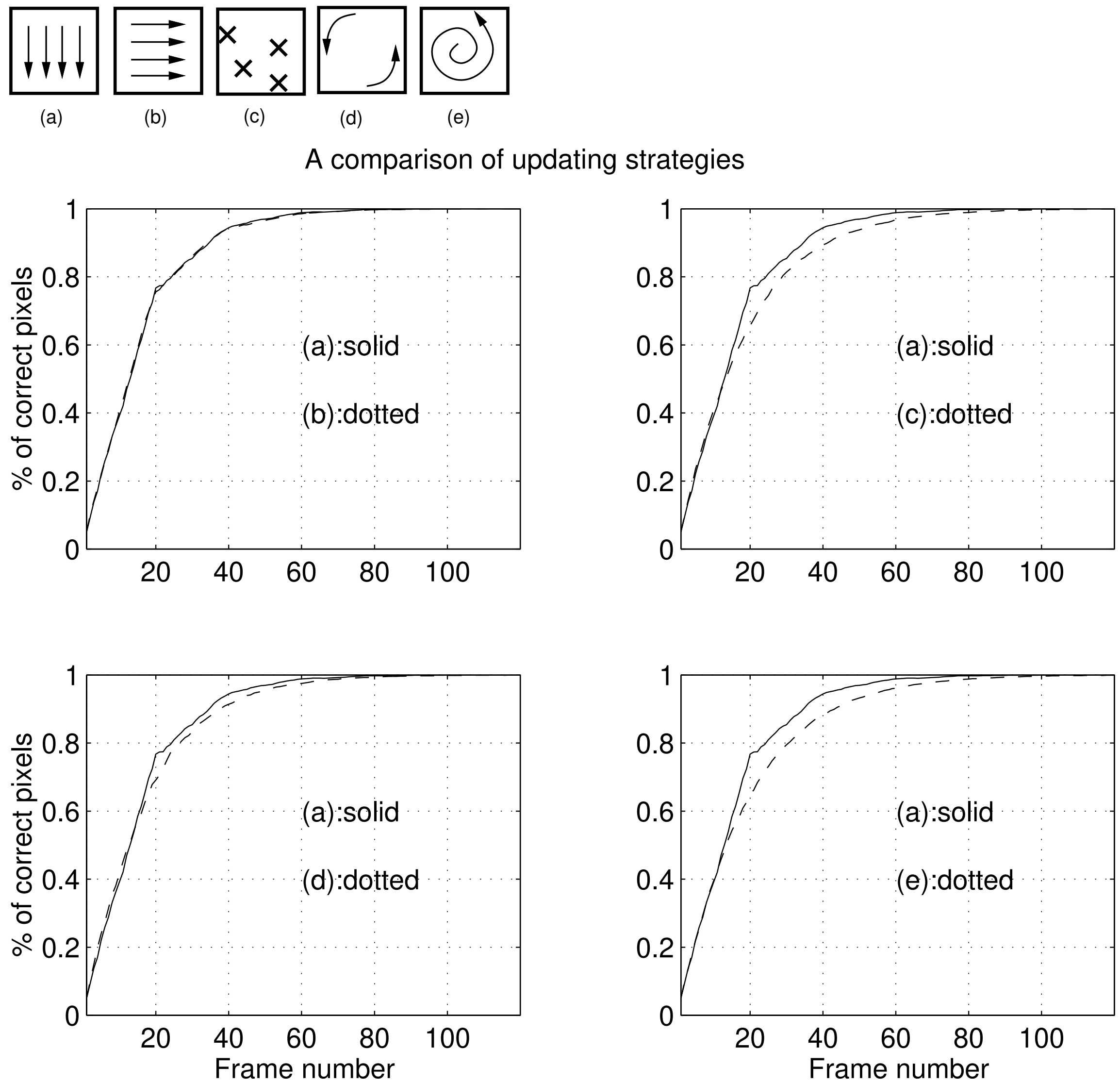


Figure 5: Static update strategies and their performance; (a) sequential in Y (b) sequential in X (c) random (d) corner (e) center

(a)

(b)

(c)

Figure 6: A comparison of the IMU scheme and static updating. The images displayed here are from the Miss America sequence. The left ones are the output of IMU and the right ones are the output of static updating. (a) 50 frames after the corruption; (b) 100 frames after the corruption; (c) 200 frames after the corruption.

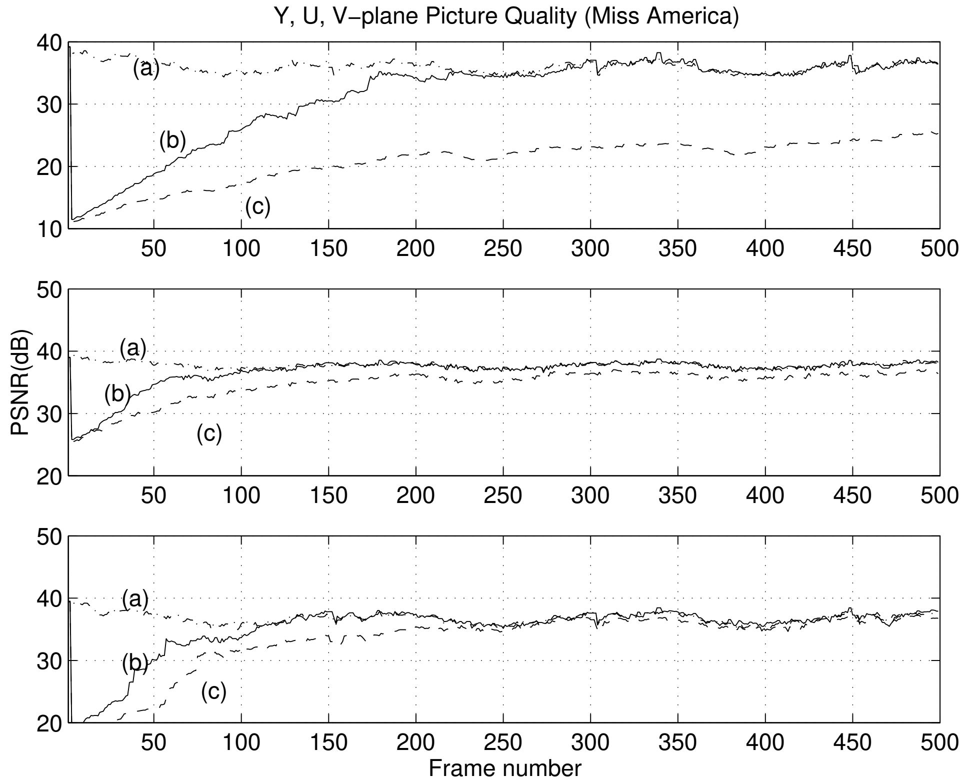


Figure 7: Error recovery curves of three sequences: (a) un-corrupted; (b) IMU; (c) static.

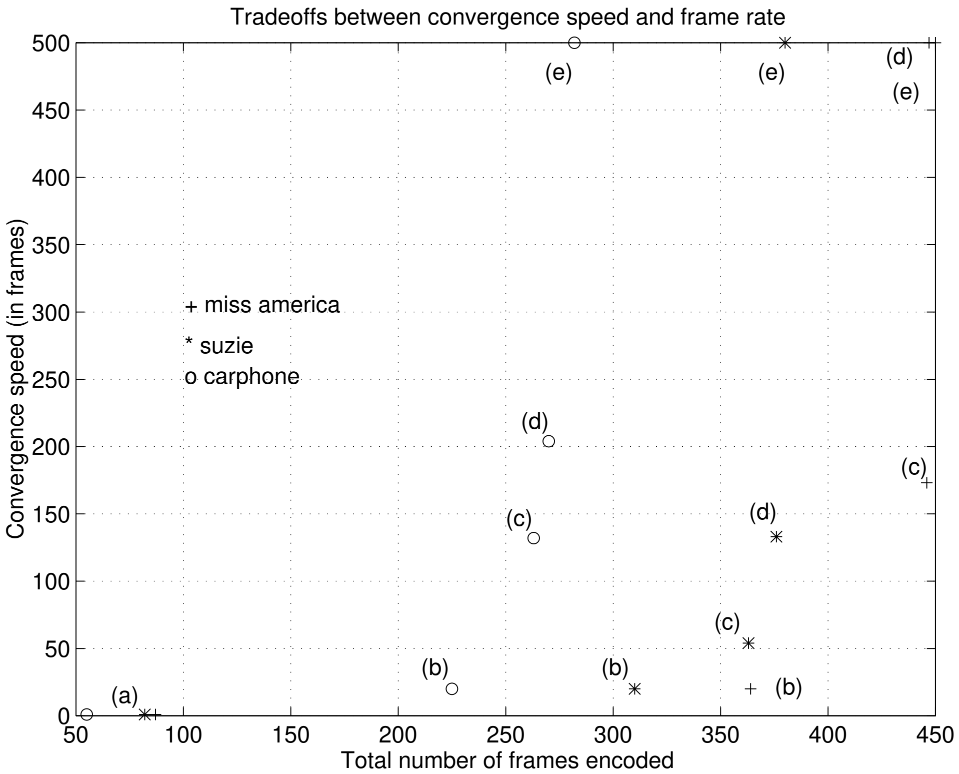


Figure 8: The tradeoffs between the convergence speed and the frame rate: (a) All I frames; (b) One I frame in 20 frames; (c) IMU; (d) static; (e) original.

that an I-frame needs more bits to encode and the rate control function is forced to drop the frame rate to keep up with the target bit rate. Scheme (b) relaxes the convergence time to 20 frames and gets approximately four times as many frames. The two schemes we are interested in comparing, the IMU scheme and the static update strategy, generate approximately the same number of frames but the IMU scheme converges 35 to 65 percent faster than the static update scheme. As pointed out in the previous section, the key idea of the IMU scheme is to rearrange the temporal and spatial locations of the force update blocks to maximize the robustness of the output stream without increasing its bandwidth. The performance evaluation results validate the effectiveness of this scheme.

5 IMU for Real-Time Encoding

For delay-critical applications, the non-real-time IMU scheme described in the previous two sections cannot be applied because video frames cannot be buffered to construct the dependence graph. The encoder thus has to rely on analyzing motion patterns in past encoded frames and predict which macroblocks are likely to be important in the future. In the following, we describe a heuristic of macroblock updating for real-time encoding.

The real-time IMU heuristic is designed based on the updating principles we discovered when observing the macroblock selections of the non-real-time IMU scheme. Based on what we learned from the simulations of updating patterns presented in Section 4, we found that an effective updating scheme obeys one or more of the following principles.

1. Unreferenced blocks are updated less often than frequently referenced blocks.
2. Macroblocks in a frame should be updated in clusters to minimize the probability of referencing corrupted regions.
3. In successive frames, updated regions should expand in such a way that minimizes the boundary between updated and corrupted regions.

Our heuristic is divided into two parts. The first part is a learning module that keeps track of the frequency of a macroblock being referenced. The learning module then adjusts the force update period (FUP) of each macroblock accordingly. The higher the reference frequency, the lower its FUP. The formula used in calculating the FUP is as follows:

$$FUP_i = FUP_{ave} - K * \arctan\left(RF_i/RF_{ave} - 1\right),$$

where K is a scaling constant. RF_i and RF_{ave} are the reference frequency of the ith macroblock and the average reference frequency of all macroblocks in a frame, respectively. FUP_{ave} is the mean value of the FUPs of all macroblocks.

The second part of the heuristic approach is a clustering module that selects which macroblocks to update. It forms an updating cluster by first selecting macroblocks that have reached their FUPs. It then examines their neighboring blocks and finds which ones are scheduled to be updated in the next few frames and reschedules their update to the current frame. Special attention is paid not to insert too many I-blocks in a single frame. Inserting too many update blocks can cause frame skipping and degrade the video quality.

Fig. 9 presents the evaluation of the real-time heuristic. In this figure, the convergence curves of the non-real-time IMU scheme and the static forced update scheme are included to provide comparisons. Our real-time heuristic scheme improves over the static scheme but is inferior to the non-real-time IMU scheme. Further research is needed to close the performance gap between the real-time and non-real-time IMUs.

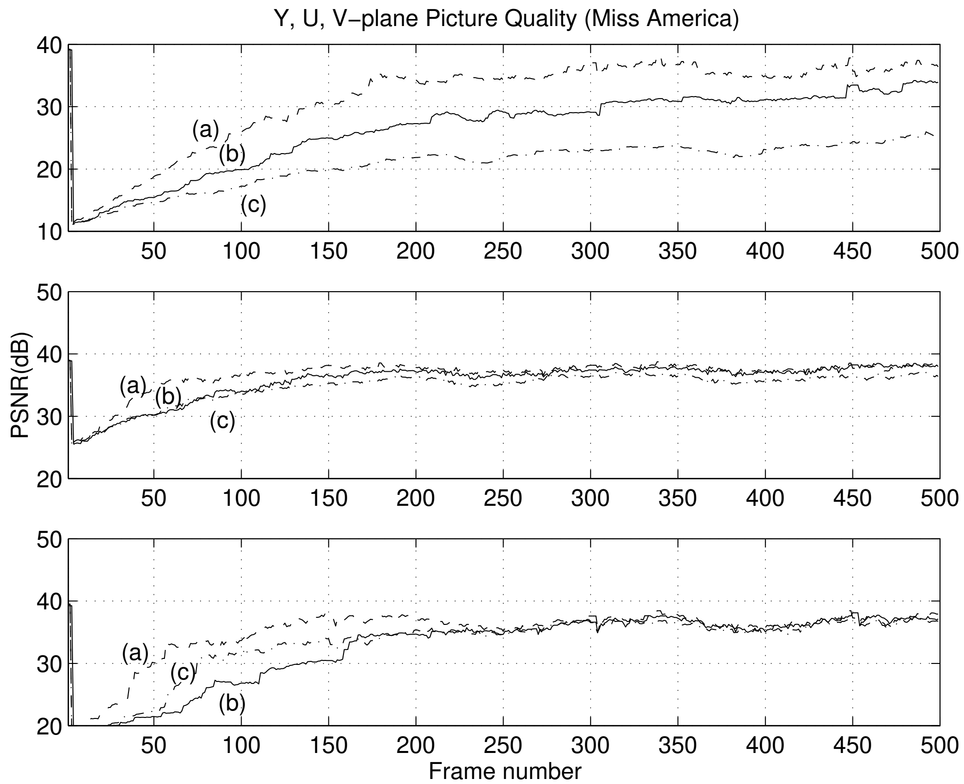


Figure 9: Error recovery curves of three sequences: (a) IMU-non-real time; (b) IMU-real time; (c) static.

6 Conclusions

A primary issue concerning the transmission of compressed video data over the Internet is how to achieve fast recovery from packet losses while maintaining a smooth bandwidth profile. We developed, and demonstrated the effectiveness of, a macroblock updating schemes which improves the error resilience of H.263 compressed video streams. The non-real-time IMU algorithm selects macroblocks to be updated by analyzing the dependencies among macroblocks in successive frames. Macroblocks having the most impact on later frames are converted to I-blocks to stop possible error propagation. The real-time IMU heuristic adaptively changes the force update periods of macroblocks depending on the frequencies with which those blocks are referenced. It also groups the force update blocks into clusters to increase the robustness. Performance evaluations of the proposed techniques demonstrate that they outperform simple static approaches and achieve a good balance between error recovery speeds and video quality.

References

[1] "Video coding for low bit-rate communication: draft ITU-T recommendation H.263," *Telecommunication Standardization Sector of ITU*, May 1996.

[2] M. W. Whybray and W. Ellis, "H.263 - video coding recommendation for PSTN videophone and multimedia," *IEE Colloquium 'Low Bit Image Coding'*, London, UK, June 6, 1995.

[3] P. Haskell and D. G. Messerschmitt, "Resynchronization of motion compensated video affected by ATM cell loss," *Proc. ICASSP-92*, San Francisco, CA, vol. 3, pp. 545-8, Mar. 1992.

[4] D. Raychaudhuri, et al., "ATM transport and cell-loss concealment techniques for MPEG video," *Proc. ICASSP-93*, New York, NY, vol. 1, pp. 117-20, Apr. 1993.

[5] A. Hung and T. Meng, "Error resilient pyramid vector quantization for image compression," *Proc. ICIP-94*, Austin, TX, vol. 1, pp. 583-7, Nov. 1994.

[6] J. Moura, et al., "Video over wireless," *IEEE Personal Communications*, vol. 3, no. 1, pp. 44-54, Feb. 1996.

[7] S. Narayannaswamy, et al., "A low-power, lightweight unit to provide ubiquitous information access application and network support for InfoPad," *IEEE Personal Communications*, vol. 3, no. 2, pp. 4-17, Apr. 1996.

[8] M. Khansari, et al., "Robust low bit rate video transmission over wireless access systems," *Proc. ICC-94*, New Orleans, LA, vol. 1, pp. 571-5, May. 1994.

[9] I. Richardson and M. Riley, "MPEG coding for error-resilient transmission," *5th International Conference on Image Processing and its Applications*, London, UK, July 1995.

[10] C. Zhu, "RTP payload for H.263," *Internet Draft*, Feb. 1996.

[11] B. Belzer, et al., "Adaptive video coding for mobile wireless networks," *Proc. ICIP-94*, Austin, TX, vol. 2, pp. 972-6, Nov. 1994.

[12] N. Naka, et al., "Improved error resilience in mobile audio visual communications," *4th International Conference on Universal Personal Communications*, Tokyo, Japan, Nov. 1995.

[13] H. Liu and M. Zarki, "Transmission of H.263 video over wireless channels," *Proc. SPIE-96*, San Jose, CA, vol. 2668, pp. 408-18, Feb. 1996.

[14] A. Alwan, et al., "Adaptive mobile multimedia networks," *IEEE Personal Communications*, vol. 3, no. 2, pp. 34-51, Apr. 1996.

[15] T. Turletti and C. Huitema, "Videoconferencing on the Internet," *IEEE Trans. on Networking*, vol. 4, no. 3, pp. 340-51, June 1996.

A Proof of the convergence property of incremental macroblock updating

Given motion vectors are stationary and uniformly distributed, prove that using the incremental updating strategy guarantees the convergence of the pixels to their correct values.

Definition:

Let P_t be a two dimensional array of size M by N with the values of its elements falling in between zero and one. The subscript t stands for the values at time t. A physical interpretation of P_t is that each of its elements represents the probability of a pixel value being correct, with probabilities equal to one as being correct and probabilities equal to zero as being incorrect.

Let R_t be a two dimensional array with the same size as P_t. Again, the subscript t stands for the values at time t. The value of an element in R_t can only be zero or one. A physical interpretation of R_t is that an element with a value of one represents its corresponding pixel getting updated. An element with a value equal to zero means its corresponding pixel is not updated.

Let $E_w(*)$ be a mapping of a two dimensional array of size M by N to another two dimensional array of the same size. We define the operation of $E_w(*)$ to be the windowing average of each element. If an element outside the array was referenced, the value of its mirrored position inside the array would be used. For example,

$$P_t(-M,-N) = P_t(M-1, N-1) for M > 0, N > 0$$

A physical interpretation of the mapping $E_w(*)$ is that it represents the computation of the expected value of the probability of a pixel being correct given the assumption that motion vectors are uniformly distributed. The size of the window is determined by the size of the motion vector search range.

Let I be an all 1's array.

The incremental updating of macroblocks can be described by the following equations:

$$P_{t+1} = \max\left(E_w(P_t), R_t\right) \tag{1}$$

,where the operation is element-wise.

$$\forall t_0, \sum_{t=t_0}^{t_0+T} R_t \geq 1 \tag{2}$$

,where T is the updating period. The first equation represents the macroblock updating operation, which either takes the predicted value, $E_w(P_t)$ or the I-block updating, R_t. The second one states that in a finite updating period T, each pixel has to be updated at least once.

¿From the definition of $E_w(*)$, it can be shown to have the following property:

$$sum(E_w(P_t)) = sum(P_t) \tag{3}$$

,where $sum(*)$ is the arithmetic sum of all elements. Since for each element in P_t, it is always less than or equal to one. We can conclude that the above summation is always less than or equal to $M * N$.

Proof:

Now we want to prove that for each element,

$$\lim_{t\to\infty} P_t = I$$

By Eqs.1 and 3, we have

$$sum(P_{t+1}) = sum(\max(E_w(P_t), R_t)) \geq sum(E_w(P_t)) = sum(P_t) \tag{4}$$

since $sum(P_{t+1}) \geq sum(P_t)$, the sequence $sum(P_t)$ is non-decreasing. For $P_t \neq I$, using Eqs.2 and 4, we have

$$\exists i, 0 \leq i \leq T-1, sum(P_t) \leq sum(P_{t+i}) < sum(P_{t+i+1}) \leq sum(P_{t+T}) \tag{5}$$

The above inequality holds because one can always find some (m, n) pairs such that

$$R_{t+i}(m,n) = 1 \; and \; E_w(P_{t+i})(m,n) < 1$$

Since $sum(P_t) < sum(P_{t+T})$ for $P_t \neq I$, $sum(P_{t+nT})$ is an increasing sequence bounded above by $M * N$. There are $M * N$ elements, each of which is again bounded above by one. Thus, it can be shown that for any t,

$$\lim_{n\to\infty} sum(P_{t+nT}) = M * N \; and \; \lim_{n\to\infty} P_{t+nT} = I$$

We conclude that $\lim_{t\to\infty} P_t = I$. The value of each element in P_t will approach one and its corresponding pixel will have a correct value.