NAIVE – Network Aware Internet Video Encoding

Hector Briceño
MIT
hbriceno@lcs.mit.edu

Steven Gortler
Harvard University
sjg@cs.harvard.edu

Leonard McMillan
MIT
mcmillan@lcs.mit.edu

Abstract

The distribution of digital video content over computer networks has become commonplace. Unfortunately, most digital video encoding standards do not degrade gracefully in the face of packet losses, which often occur in a bursty fashion. We propose an new video encoding system that scales well with respect to the network's performance and degrades gracefully under packet loss. Our encoder sends packets that consist of a small random subset of pixels distributed throughout a video frame. The receiver places samples in their proper location (through a previously agreed ordering), and applies a reconstruction algorithm on the received samples to produce an image. Each of the packets is independent, and does not depend on the successful transmission of any other packets. Also, each packet contains information that is distributed over the entire image. We also apply spatial and temporal optimization to achieve better compression.

1 Introduction

With the advent of the internet, the distribution of digital video content over computer networks has become commonplace. Unfortunately, digital video standards were not designed to be used on computer networks. Instead, they generally assume a fixed bandwidth and reliable transport from the sender to the receiver. However, for the typical user, the internet does not make any such guarantees about bandwidth, latency or errors. This has lead to the adaptation or repackaging of existing video encoding standards to meet these constraints. These attempts have met with varying levels of success. In this paper we propose to design a new video standard specifically for computer networks from the ground up.

The internet is a heterogeneous network whose basic unit of transmission is a packet. In order to assure scalability, the internet was designed as a best effort network - i.e. it makes no guarantees that a packet sent by a host will arrive at the receiver or that it will be delivered in the order that it was sent. This also implies that it makes no guarantees on the latency of the delivery.

A video encoding system designed for computer networks would ideally satisfy the following requirements. The transmitted data stream should be tolerant to variations in bandwidth and error rates along various networking routing paths. A given data stream should also be capable of supporting different qualities of service. Where this quality of service might be dictated by local resources (such as CPU performance) or the other user requirements. These requirements are only partially satisfied by existing video encoding systems. In this paper we propose a flexible video encoding system that satisfies the following design goals:

- The system must allow for broadcast. We would like a system where video can be transmitted to a large audience in real time with no feedback to the source. This allows for arbitrary scalability.
- The network can arbitrarily drop packets due to congestion or difference of bandwidths between networks or receivers. Since this system is targeted to error prone networks, it must perform well under packet losses.
- The sender should be able to dynamically vary the bandwidth and CPU requirements of the encoding algorithm. In order to guarantee a quality of service variations in bandwidth may be necessary. For instance, at scene changes or during a complex sequence. Variations in bandwidth could also occur due to resource limitations at the source such as

channel capacity and CPU utilization, or by a policy decision.

- The receiver should be able construct a reasonable approximation of the desired stream using a subset of the data transmitted. Furthermore, the receiver may also intentionally ignore part of the data received to free up resources in exchange for reduced quality.
- The quality of the video should degrade gracefully under packet loss by the network or throttling by the sender or receiver.
- Variations in the algorithm should support a wide range of performance levels, from small personal appliances to high-end workstations.
- User should be able to quickly join a session in progress.

These goals place severe constraints on how the system can be built.

Packets are the basic unit of network transmission we consider [13]. A video frame generally spans many packets. System throughput and quality are affected by throttling packets at the sender, packet loss in the network, and ignoring of packets at the receiver. Therefore, we choose to regard packets as atomic in our system design. For scalability and error handling we avoid packets that contain prioritized data or interdependencies, such as the clustering of data or differential encoding. These goals motivate our design principles:

Globalness – Individual packets should contain enough information to reconstruct the whole image. They also should be additive - each additional packet increases the reconstructed image quality. Conversely, for each packet that is dropped by the sender, network or receiver, the image quality degrades.

Independence – All packets are independent of each other; any one of them can be dropped without abrupt changes in quality, and in many cases we can process them out of order.

These principles are quite different than current video encoding systems. Typical video encoding algorithms (i.e. H.263 [1] or ISO MPEG), use compression and encoding techniques that make packets interdependent; when one packet is lost, all other packets that are related to it lose their usefulness.

We propose an encoding system that scales well with respect to the sender's performance, the number of receivers, and the network's performance. This system degrades gracefully under packet loss. Briefly stated: the encoder sends packets that consist of a small random subset of pixels distributed throughout a video frame. The receiver places samples in their proper location (through a previously agreed ordering), and applies a reconstruction algorithm on these samples to produce an image. Notice that since each packet contains a small random subset of the image, there is no ordering or priority for packets. We also apply spatial and temporal optimization to achieve better compression without compromising our global and independence principles.

2 Previous Work

Video encoding algorithms specifically tailored for the internet have been previously proposed. ISO MPEG-1 provides high compression ratios, and it allows for bitstream resynchronization using slices. Generally slices span multiple packets, and few encoders make an effort to align slices within packet boundaries. The variable length encoding and difference encoding used by MPEG-1 is very effective in reducing the bitrate, but both techniques make assumptions about what has been previously received. If these assumptions are wrong (caused by packet loss) [8], artifacts will develop in the new frame. Other discrete cosine transform (DCT) based algorithms like H.261, have been successfully adapted for use in computer networks by using a technique sometimes called "conditional replenishment" [19]. The idea is, that instead of encoding the differences from previous frames, they either keep old blocks or entirely replenish new blocks independently encoded. These techniques require that all blocks are replenished within a specified period of time. During heavy packet losses, important areas may not be updated until the losses subside. This is an all or nothing approach: a block will completely reach its new state or not change at all.

Layering approaches have partly alleviated this last problem. Algorithms like L-DCT [2] and PVH [19], use a base channel to encode a low quality representation of the block; and use additional channels to encode enhancement information to reproduce a more faithful block. Because enhancement layers usually depend on the base layered being received, when the base layer packets are lost, the block cannot be updated at all.

Error handling can also be incorporated into the network layer. By using error correcting codes, or retransmission based schemes, errors can be minimized or eliminated, as to create the illusion of a reliable network stream. Open-loop approaches [28] (i.e. those that don't require feedback) such as, Forward Error Correction (FEC), eliminate errors when they are well characterized. Unfortunately, these systems must include enough redundancy in advance to deal with the worst-case packet loss rate scenario. This leads to inefficiencies. The overhead for error correction also increases total network load. Thus the entire network is taxed due to the worse performing route [23, 12]. The alternative is to use a closed-loop approach. Close-loop approaches [25, 22, 7, 29] , where the receivers request the retransmission of lost packets, have the drawback of higher latency and are difficult to scale [6, 4]. Additionally, since packet loses generally occur during congestion, these requests and subsequent retransmissions can make matters worse.

The algorithm we propose bears many resemblences to work in error concealment [3, 11, 30, 27]. While most error concealment techniques are built upon existing standards, our technique proposes an entirely novel encoding scheme. Our encoding scheme is tolerant to bursty errors, and does not require resynchronization. Our reconstruction algorithm is fast, and makes no a-priori assumptions about the existance of specific nearby blocks or pixels.

3 The Algorithm

The Network Aware Internet Video Encoding (NAIVE) system sends a small random subset of samples from each video frame and reconstructs the frame at the receiver. The random samples can be distributed across one or more network packets. Randomness is used to select samples in order to decorrelate errors and reduce artifacts such as blockiness. Following our design principles, each packet contains samples uniformly distributed throughout the whole image, and independent of any previous packet sent. Our encoding system allows for arbitrary packet loss, thus there is no guarantee that the client has received any particular set of image information. This presents us with the problem of reconstructing an image from irregularly spaced samples.

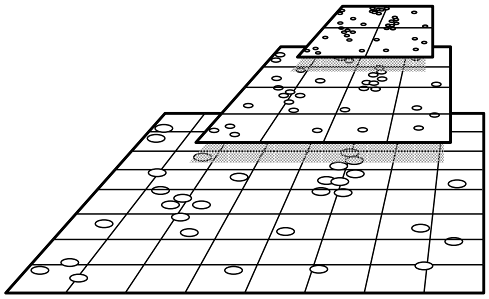


Figure 1: 2D pull-push. At lower resolutions the gaps are smaller.

3.1 Image Reconstruction

A viable solution to this image reconstruction problem must have the following features:

- The method must run at frame rate. Thus, it is too expensive to solve systems of equations (as is done when using global spline methods [26, 17]) or to build spatial data structures (such as a Delauney triangulation [21]).
- The method must deal with spatially scattered samples. Thus we are unable to use standard interpolation methods, or Fourier-based sampling theory.
- The method must create reconstructions of acceptable quality.

In this paper we adapt the pull-push algorithm of Gortler et al. [14]. This algorithm is based on concepts from image pyramids [9], wavelets [18] and subband coding [16], and it extends earlier ideas found in [10] and [20]. The algorithm proceeds in two phases called pull and push. During the first phase, pull, a hierarchical set of lower resolution data sets is created in an image pyramid. Each of these lower resolution images represents a "blurred" version of the input data; at lower resolutions, the gaps in the data become smaller (see figure 1). During the second phase, push, this low resolution data is used to fill in the gaps at the higher resolutions. Care is taken not to destroy high resolution information where it

Figure 2: Grayscale lenna image samples and reconstruction. Using 22% original pixels (a), and using 5% of original pixels (b). The images in the left column show the input pixels. The right column shows our reconstruction

is available. Figure 2 shows the reconstruction of the lenna grayscale from 5% and 22% of the original pixels.

3.1.1 Organization

The algorithm uses a hierarchical set of image pixels with the highest resolution labeled 0, and lower resolutions having higher indices. Each resolution has 1/2 the resolution in both the horizontal and vertical dimensions. For our 320 by 240 images, we use a 5 level pyramid. Associated with the ij'th pixel value $p^r_{i,j}$ at resolution r is a weight $w^r_{i,j}$. These weights, representing pixel confidence, determine how the pixels at different resolution levels are eventually combined.

3.1.2 Initialize

During initialization, each of the received pixels is used to set the associated pixel value $p^0_{i,j}$ in the high resolution image, and the associated weight $w^0_{i,j}$ for this pixel is set to f. f is the value chosen to represent full confidence. The meaning of f is discussed below. All other weights at the high resolution are set to 0.

3.1.3 Pull

The pull phase is applied hierarchically, starting from the highest resolution and going until the lowest resolution in the image pyramid. In this pull phase, successive lower resolution approximations of the image are derived from the adjacent higher resolution by performing a convolution with a discrete low pass filter $\tilde{h}$. In our system, we use the "tent" sequence. $\tilde{h}[-1..1] \times [-1..1]$:

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

The lower resolution pixels are computed by combining the higher resolution pixels using $\tilde{h}$. One way to do this would be to compute

$$\begin{array}{lll} w^{r+1}_{i,j} & := & \sum_{k,l} \tilde{h}_{k-2i,l-2i}\, w^r_{k,l} \\ p^{r+1}_{i,j} & := \frac{1}{w^{r+1}_{i,j}} & \sum_{k,l} \tilde{h}_{k-2i,l-2j}\, w^r_{k,l}\, p^r_{k,l} \end{array} \tag{1}$$

This is equivalent to convolving with $\tilde{h}$ and then downsampling by a factor of two.

This computation can be interpreted as follows: Suppose we have a set of continuous tent filter functions associated with each pixel in the image pyramid. Suppose $\tilde{B}^0_{i,j}(u,v)$ is a continuous piecewise bilinear linear tent function centered at i,j and two units (high resolution pixels) wide, $\tilde{B}^1_{i,j}(u,v)$ at the next lower resolution is a tent function centered at $2i, 2j$ and is four units (high resolution pixels) wide, $\tilde{B}^2_{i,j}(u,v)$ at the next lower resolution is a tent function centered at $4i, 4j$ and is 8 units wide, and so on. These continuous functions are related using the discrete sequence $\tilde{h}$:

$$\tilde{B}^{r+1}_{i,j}(u,v) = \sum_{k,l} \tilde{h}_{k-2i,l-2i} \tilde{B}^r_{k,l}(u,v)$$

This means that one can linearly combine finer tents to obtain a lower resolution tent. The desired multiresolution pixel values can be expressed as an integral over an original continuous image $P(u,v)$ using the $\tilde{B}^r_{i,j}(u,v)$ as weighting functions:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} du\ dv\ \ \tilde{B}^r_{i,j}(u,v)P(u,v) \tag{2}$$

If one approximates this integral with a discrete sum over the received pixel values, one obtains

$$w^r_{i,j} p^r_{i,j} = \sum_{k,l} \tilde{B}^r_{i,j}(k,l) p^0_{k,l} w^0_{k,l} \tag{3}$$

where

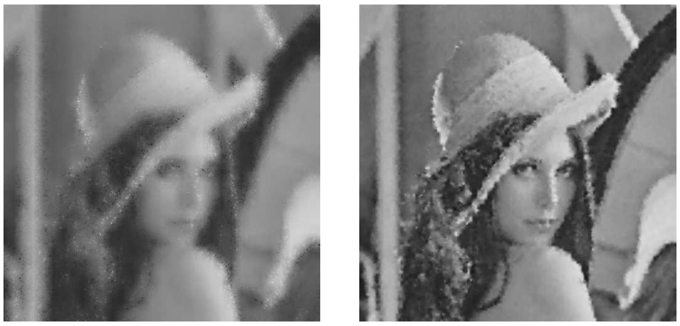
$$w^r_{i,j} = \sum_{k,l} \tilde{B}^r_{i,j}(k,l) w^0_{k,l}$$

It is easy to show that the values computed by Equation 3 can be exactly and efficiently obtained by applying Equation 1 hierarchically.

This method creates good low resolution images when the original samples are uniformly distributed. But when the original samples are unevenly distributed, Equation 3 becomes a biased estimator of the desired low resolution value defined by Equation 2 for it overly emphasizes the over sampled regions. Our solution to this problem is to replace Equation 1 with:

$$\begin{array}{lcll} w^{r+1}_{i,j} & := & & \sum_{k,l} \tilde{h}_{k-2i,l-2i} \min(w^r_{k,l}, f) \\ p^{r+1}_{i,j} & := & \frac{1}{w^{r+1}_{i,j}} & \sum_{k,l} \tilde{h}_{k-2i,l-2j} \min(w^r_{k,l}, f)\, p^r_{k,l} \end{array} \tag{4}$$

The value f represents full confidence, and the $\min$ operator is used to place an upper bound on the degree that one image pyramid pixel corresponding to a highly sampled region, can influence the total sum. Any value of $1/16 \leq f \leq 1$ creates a well defined algorithm. If f is set to one, then no saturation is applied, and this equation is equivalent to Equation 1. If f is set to $1/16$, then even a single sample under the sum is enough to saturate the computation for the next lower resolution. In the system we have experimented with many values, and have obtained the best results with $f = 1/8$. Although complete theoretical analysis of the estimator in Equation 4 has yet to be completed, our experiments show it to be far superior to Equation 1. Figure 3 shows the reconstruction of the lenna grayscale image with 10% of its samples reconstructed using (a) f = 1, (b) f = 1/8.



(a) (b)

Figure 3: Grayscale lenna test image reconstruction with 10% of samples: (a) using f = 1, (b) f = 1/8

The pull stage runs in time linear in the number of pixels summed over all of the resolutions. Because each lower resolution has half the density of pixels, the computation time can be expressed as a geometric series and thus this stage runs in time linear in the number of high resolution pixels at resolution 0.

3.1.4 Push

The push phase is also applied hierarchically, starting from the lowest resolution in the image pyramid, and working to the highest resolution. During the push stage, low resolution approximations are used to fill in the regions that have low confidence in the higher resolution images. If a higher resolution pixel has a high associated confidence (i.e., has weight greater than or equal to f), we disregard the lower resolution information for that high resolution pixel. If the higher resolution pixel does not have sufficient weight, we blend in the information from the lower resolution.

To blend this information, the low resolution approximation of the function must be expressed in the higher resolution. This is done using an interpolation sequence also based on the tent sequence but with a different normalization: $h[-1..1] \times [-1..1]$:

$$\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

Push proceeds is done in two steps: we first compute temporary values

$$tp^r_{i,j} \quad := \quad \sum_{k,l} h_{i-2k,j-2l}\, p^{r+1}_{k,l}$$

This computation is equivalent to upsampling by a factor of 2 (adding 0 values), and then convolving with h. These temporary values are now ready to be blended with the p^r values already at level r, using the w^r as the blending factors.

$$p^r_{i,j} \quad := \quad (1 - \frac{w^r_{i,j}}{f})\, tp^r_{i,j} + \frac{w^r_{i,j}}{f}\, p^r_{i,j}$$

analogous to the "over" blending performed in image compositing [24].

3.1.5 Lower Resolution Samples

There can be cases in which the sender wishes to send only low resolution information about some image region (perhaps that region is blurry or it is deemed to be less important). Our algorithm allows the sender to send lower resolution pixels directly to the appropriate level of the image pyramid, $p^r_{i,j}$ for $r > 0$. When such pixels are received they are placed directly in the image pyramid at the appropriate resolution, and we suppress the pulling of high resolution pixels to it. This allows the sender to avoid sending many high resolution pixels where the information content is primarily of low frequency.

3.1.6 Temporal Coherence

In video sequences, image regions can change slowly. Our system takes advantage of this temporal coherence by allowing pixels from previous frames to be included in the pull-push reconstruction process.

3.2 Packetization

The pull-push algorithm provides a means of reconstructing an image from non-uniform samples. From our principle of globalness we need to pick samples from the whole image. And these have to be selected at random to avoid visible artifacts and to allow the appearance of simultaneous update everywhere in the image [5]. We guarantee coverage of the whole image by dividing it into 16x16 blocks and making succesive passes over the image selecting one random sample from each block on each pass.

In order to minimize the information transmitted, the sender and the receiver agree on the ordering of samples, such that the sender only needs to send the location of the first sample in a packet. This is done as follows. The image is split into 16x16 blocks, this means that there are 256 samples per block. Say there are N blocks in an image. We generate a table, called the "offset table", that has 256*N entries. The first entry contains the coordinate of a random sample in the first block; the second entry contains the coordinate of a sample in the second block; The N+1th entry contains the location of a sample

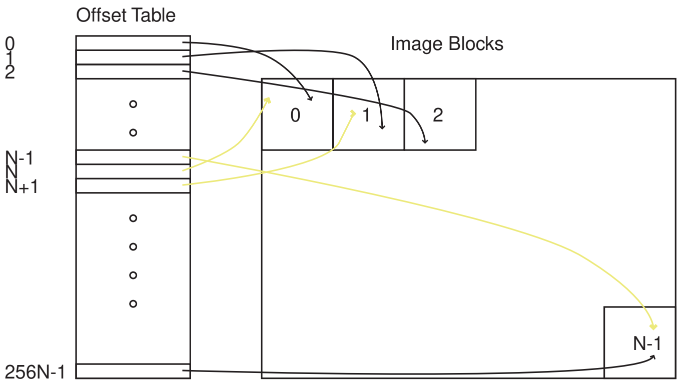


Figure 4: Offset Table: There are N 16x16 blocks in the image. The i'th entry points to a sample in block number i modulo N. On any selection of N consecutive entries, there is a sample from every block

again in the first block. The random ordering of the samples within a block is established by assigning a pseudo-random number to each pixel. The pixels are then sorted into a list according to this random number. The offset table can then be constructed by selecting a pixel from each of the N lists. The sender and receiver are synchronized through the transmission of a seed for the random number generator. This seed must be transmitted via a reliable protocol such as TCP/IP.

This ordering guarantees that if we pick N consecutive samples, they will span the whole image without large clusters. Additionally, we can easily compute the block that a sample belongs to, by computing the module N of its location in the table. See figure 4.

The reconstruction explained so far applies to a grayscale image. This same idea can be extended to the chrominance components of color images. We encode color images by sampling the chrominance components at a resolution 1/4 of the luminance image, similar to MPEG. To encode them, we maintain another offset table with 8x8 blocks to correspond to the 16x16 blocks of the luminance components. We encode the chrominance samples independently of the luminance samples.

We need to send very little overhead information with each packet. Each packet consists of: a frame number; table offset of first chrominance sample, number of chrominance samples, and the samples themselves; and table offset of first luminance sample, with the remaining

Frame Number	#UV samples	Offset UV samples	Offset Y samples	UV samples	Y samples

Figure 5: Packet Format

of the packet filled with luminance samples. We use 1024 bytes as our default packet size. This structure satisfies our global and independence properties. If a packet has more than N luminance samples (where N is the number of blocks in a frame), then there will be one sample in every block of the image guaranteed by the way we traverse the offset table.

4 Enhancements

The baseline approach described above works well for images whose details are uniformly distributed throughout the whole image. Most images, though, have localized regions of detail. And most sequences bear a high level of temporal coherency across frames. We can take advantages of these characteristics to produce better quality video with the same or less amount of data.

4.1 Spatial Locality

In image regions with mostly low frequency content, our encoding system allows us to directly transmit lower resolution samples, and the receiver can insert these directly into lower resolution pyramid levels.

In our encoding system, we encode the sample value and resolution level in the same byte. We use 7 bits of precision for level 0 samples, and 6 bits of precision for level 1 and level 2 samples. If the least significant bit is 0, the sample is a level 0 sample; if the least significant bits is 01 or 11 the sample is a level 1 or level 2 sample respectively. With this change we keep the packet structure unchanged, except for how sample values are interpreted.

Samples that are inserted at lower resolution levels, correspond spatially to many more samples at finer levels. Thus, when a low resolution sample is sent, fewer higher resolution samples are needed.

To manage the bookkeeping for this information, we use a special table, called the SKIP TABLE. There is a SKIP TABLE entry for each block. The SKIP TABLE contains the encoder/decoder agreed upon number of samples for this block that will be skipped. When a packet is received, all entries in the SKIP TABLE are initialized to 0; thus each block is guaranteed to have one sample. When a sample is inserted into a lower resolution level, we load the skip table entry for that block, with a predefined constant, agreed upon by the sender and the receiver. In our system, when a sample is sent for level 1, we skip the next 3 samples for this block. When a sample is sent for level 2, we skip the next 15 samples for this block.

Each time that block occurs in the sequence we inspect the skip table entry to see if it is non-zero, if it is, we decrement the skip table, and go to the next block without reading a sample from the packet. Otherwise, we insert the current sample into the block according to the offset table entry.

4.2 Temporal Locality

Temporal locality can be exploited even when packets are independent of each other. MPEG and H.261 exploit temporal locality by reusing block of pixels that are closely located in the previous frame, encoding this location and their difference. In our approach, we don't make any assumptions about the previous frame or what packets the receiver has processed. We simply take advantage of the fact that pixels in a block may not change significantly across many frames, in which case, we reuse them to reconstruct a higher quality image. In NAIVE, pixels from previous frames can be kept around for up to 20 frames, and used as equal participants in the pull-push algorithm. When a block has changed significantly, a KILL_BLOCK signal is sent for that block, and all pixels for that block from previous frames are discarded. For scene changes, a KILL_ALL_BLOCKS signal will discard all previous pixels from previous frames.

We flush the previous frame samples for a given block by using a special word (KILL_BLOCK) instead of encoding the sample. When this code is seen, the block that corresponds to the offset for that sample, will be marked, and all corresponding samples from previous frames are flushed. Additionally, we do not increment the pointer into the offset table, such that the next sample in the stream falls in the current block.

Blocks that do not change will slowly improve in

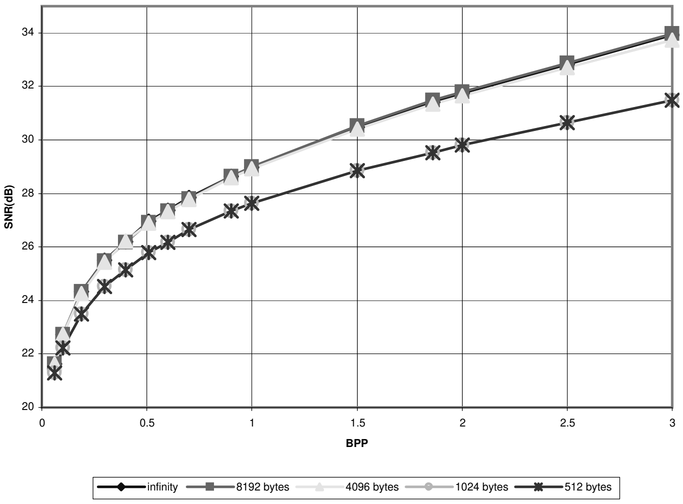


Figure 6: Rate-distortion curve on the grayscale 512x512 "Lena" test image.

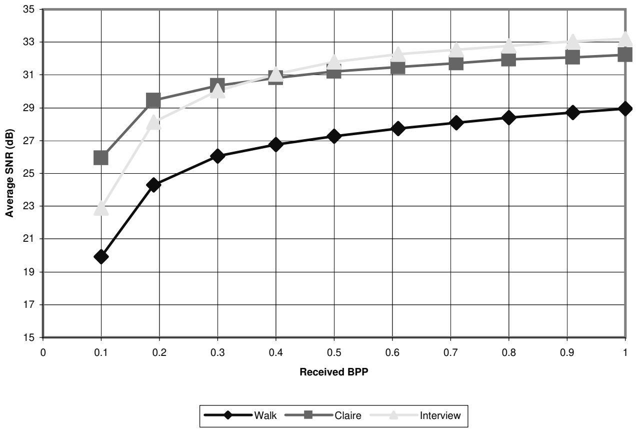


Figure 7: Average SNR of 3 color sequences with 100 frames encoded at 1 bpp [2] and decoded with different packet drop rates yielding different bpp. receive rates.

quality because they are reusing samples from previous frames; therefore we wish to use more samples to the blocks which are changing more rapidly and are not reusing samples. We accomplish this by inserting negative values in the SKIP TABLE in the following way. When a block is killed, we set its corresponding SKIP TABLE entry to a negative value (currently -10). After we have gone around once for all blocks in the image, we only visit blocks that have a negative SKIP TABLE entry and increment its SKIP TABLE for each sample received. This continues until there are no more negative SKIP TABLE entries left. This increases the reconstructed quality of blocks that are not reusing previous samples. This does not violate our globalness principle, since we still have at least one sample per every block if they fit in a packet.

5 Results

In this section we evaluate the performance of our compression system. Before we proceed it is important to note two caveats. First, the policies of the encoder will greatly determine the quality of the decompressed stream. The encoder can make many decisions. For example, it can make decisions about which blocks to flush or keep, what offset to start sending samples from, from which levels samples should be drawn, what proportion of luminance/ chrominance samples to use, among others decisions. We have manually found reasonable settings for our video streams. In the optimal case, the encoder would make these decisions automatically. Secondly, we have used the signal-to-noise ratio metric (SNR) for evaluating our results. It is well known that SNR is not an optimal measurement for image quality. It is acceptable for comparing the algorithms based on the same transform with different settings [15]. A better measurement would be based on models of the human visual system; but these are usually harder to implement or compute than the SNR.

Figure 6 shows the rate distortion curve for 512x512 grayscale image, compressed for different target bit per pixels (bpp) and different packet sizes. Large packet sizes are important for large images. If the packet is not larger than the number of blocks in an image, then there will not be enough space to go around all the blocks once, and more importantly, the algorithm will not make use of the SKIP TABLE, which allows it to get more samples in needed areas. The drawback of using large packets is that they are more likely to fragmented and lost. When a packet is fragmented, and one of its fragments get lost, the whole packet is lost. For small images, a packet size of 1024 bytes is adequate. For our experiments we used a packet size of 1024 bytes because it is compatible with the maximum packet size of most networks.

Figure 7 shows how the quality degrades gracefully

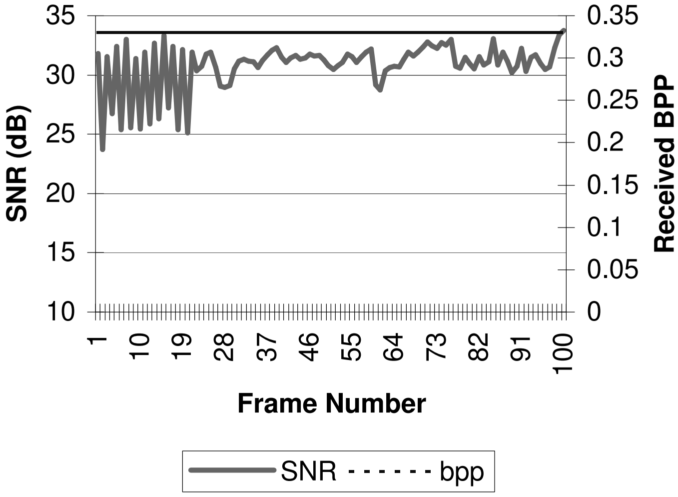


Figure 8: Base: SNR for each frame given the bpp received per frame, constant receive rate of 0.33 bpp

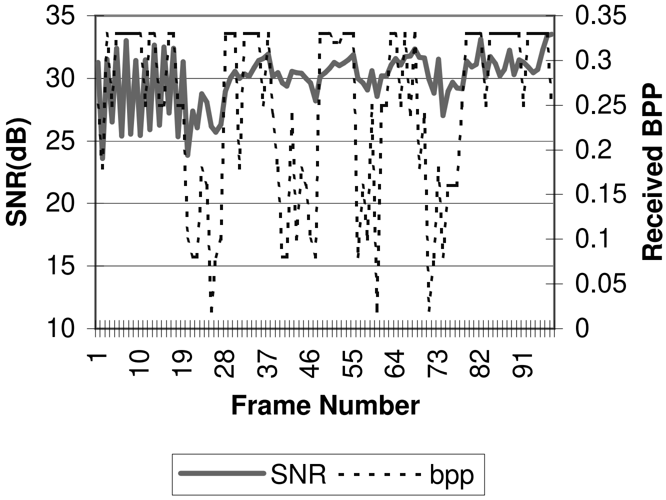


Figure 9: Bursty: SNR for each frame given the bpp received per frame, there are bursty errors, so the receive rate drops sporadically

for different kinds of video sequences. For these sequences, temporal and spacial locality has been used. The first sequence, *Walk*, contains a men in suits walking from a car, the scene has high detail and motion. The second sequence, *Claire* is a standard head and shoulders shot. Lastly, the *Interview*, consists of three scenes: a person walking into a room, a head and shoulders shot of the person talking inside the room, and close up of her face. All three sequences contain 100 frames, and were encoded at 1bpp. To generate all the data, the sequences were decoded with different packet drop rates calculating the average SNR of all frames. The packet drop rate determines the independent probability that a packet will be dropped. Over a whole sequence, a video encoded at 1bpp and decoded with a packet drop rate of 30%, will have a receive bpp of 0.7bpp. The slope of all three curves is very similar, showing that it degrades slowly regardless of the kind of video.

The algorithm handles bursty packet losses well. Figure 8 shows the frame by frame SNR for a 10 second interview (320x240 color) sequence compressed at 0.33 bpp. This sequence is composed of three shots. The first 22 frames is a shot sequence of the person walking into an office. The stride of the person and camera angle makes the shot contain one slow motion frame and one fast motion frame, to give the resulting wave-like shape for the SNR during that shot. The second shot is a head and shoulders shot of the person being interview in her office. This shot lasts until frame 77. The last shot is a close up of the person. The quality of the image is above 30dB for most of the sequence, there is a short dip between frame 77 and frame 78, but it does not take long to recover.

Figure 9 shows the same sequence under bursty packet loss. The dashed line represents the actual bit rate during the reception of each frame. This figure shows that even under heavy loss (receiving less that 0.1 bpp), the quality does not degrade significantly. At the end of the first burst, in frame 28, the quality level recovers rapidly. Additionally, the quality hardly degrades during the second burst, between frames 37 and 47.

The complexity of the algorithm is simple enough to allow a software-only implementation. Table 1 shows the decoding frame rate for different sequences. The algorithm was run on a common Intel Pentium Pro 200Mhz processor running Linux and the X windows system. The frame rate is not very sensitive to the amount of data received. The decoding time is dominated by the pull-push algorithm after all the samples received from the network have been placed in the image. The color sequence ran at 50% lower frame rate, than the comparable gray scale sequence. This makes sense, since we have to reconstruct the chrominance data which is half the size of the

Test Sequence	fps 1bpp	fps 0.5bpp
interview (color 320x240)	23.5	25.3
susie (gray 352x240)	34.81	36.32
qclaire (gray 176x144)	76.7	84.9

Table 1: Decoding frame rates (without displaying) for different sequences.

luminance data for color sequences. Displaying QCIF sequences in real time would not be a problem, and with a faster machine and an efficient display system, the same might be possible for CIF sequences.

6 Conclusions

The NAIVE system that we have presented is an initial step towards a video compression system tailored specifically for computer networking environments. NAIVE satisfies our initial design goals. It supports broadcast over large-area network and maintains scalability. NAIVE is tolerant to packet loss at any point along the network from the sender to the receiver. In fact, the intentional dropping of packets at the source is one method of increasing the effective compression of the bit stream. Similarly, the selective dropping of packets at the receiver effectively sheds CPU load. A NAIVE sender can also dynamically vary its transmission bandwidth when required by the video sequence in order to maintain a given quality level. In all cases, the receiver of a NAIVE video stream is able to reconstruct a reasonable approximation of an entire frame using a minimum on information (i.e. a single packet). The reception of additional packets further enhances the quality of the frame. Finally, our system degrades gracefully under severe packet losses.

Fundamentally, the randomizing of samples used in our NAIVE method has the effect of decorrelating the input signal and effective compression methods essentially depend on highly correlated input signals. Thus, our NAIVE algorithm sacrifices compression ratio, as compared to other video compression techniques, in order to achieve our design goals. We believe that other compression techniques can be layered onto our NAIVE methods to achieve substantially improved compression. For instance, variable length encoding techniques can be applied within individual packets to reduce redundancy in the transmitted symbols. Differential encoding methods could be applied to all samples in the packet following the initial sample for each block. We are also hopeful that motion compensation techniques can be applied within our framework by encoding motion vector for each block. These motion vectors would imply that a block of samples in all pyramid levels would be copied to the current block. Thus, the sender would make no specific assumption concerning which samples are available at the receiver, only that those samples within the transferred block would form the best basis for reconstructing the desired block. It is also possible to incorporate embedded coding techniques to the samples within each packet. This would potentially allow for trading off the quantization of samples for increased sampling density.

Another shortcoming of our NAIVE method is that the sender is fundamentally unable to make any quality guarantees to any particular receiver. The need for such a guarantee might arise based from an economics driven approach where particular receivers pay a premium for assurances of a given quality level. Layering is an effective technique for satisfying such requirements. We believe that our NAIVE method could be extended to provide layering.

In summary, we view our NAIVE algorithm as starting point for the development of a new class of video compression methods that are well suited for computer networks. By considering the realities of real networks we believe that is possible to define new classes of algorithms that are scalable in broadcast applications and degrade gracefully under variations in network activity.

Acknowledgements

We would like to thank Aaron Isaksen for his help in preparing our videos. Support for this research was provided by DARPA contract N30602-97-1-0283, and Massachusetts Institute of Technology's Laboratory for Computer Science.

References

[1] H.263: Video coding for low bitrate communication. *Draft ITU-T Recommendation H.263.*, May

1996.

[2] Elen Amir, Steven McCanne, and Martin Vetterli. A layered dct coder for internet video. In *IEEE International Conference on Image Processing*, pages 13–16, Lausanne, Switzerland, September 1996.

[3] E. Asbun and E. Delp. Real-time error concealment in compressed digital video streams. *Proceedings of the Picture Coding Symposium 1999*, April 1999.

[4] Ernst W. Biersack. A performance study of foward error correction in atm networks. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSDAV) 1993*, pages 391–399, Heildelberg, Germany, November 1993.

[5] G. Bishop, H. Fuchs, L. McMillan, and E. Scher Zaiger. Frameless rendering: Double buffering considered harmful. *Computer Graphics (SIGGRAPH 94)*, pages 175–176, 1994.

[6] Jean-Chrysostome Bolot, Hugues Crepin, and Andres Vega Garcia. Analysis of audio packet loss in the internet. In *NOSDAV*, pages 154–165, Durham, NH, 1995.

[7] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. Scalable feedback control for multicast video distribution in the internet. In *ACM Communication Architectures, Protocols, and Applications (SIGCOMM) 1994*, pages 58–67, London, UK, 1994.

[8] J. Boyce and R. Gaglianello. Packet loss effects on mpeg video sent over the public internet. *ACM Multimedia, 1998*, 1998.

[9] P. Burt and E. Adelson. Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4), April 1983.

[10] P. J. Burt. Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proceedings of Computer Vision and Pattern Recognition*, pages 144–152. IEEE Computer Society Press, June 1988.

[11] Y. Chung, J. Kim, and C. Kuo. Dct based error concealment for rtsp video over a modem internet connection. *International Symposium on Circuits and Systems '98*, May 1998.

[12] I. Cidon, A. Khamisy, and M. Sidi. Analysis of packet loss processes in high-speed networks. *IEEE Trans. Info. Theory*, 39(1), January 1993.

[13] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM Communication Architectures, Protocols, and Applications (SIGCOMM) 1990*, September 1990.

[14] S. Gortler, R. Grzeszczuk, and M. Cohen R. Szeliski. The lumigraph. *Computer Graphics (SIGGRAPH 96)*, pages 43–54, 1996.

[15] Yung-Kai Lai, Jin Li, and C.-C. Jay Kuo. A wavelet approach to compressed image quality measurement. *30th Annual Asilomar Conference on Signals, Systems, and Computers*, November 1996.

[16] A. Lippman and W.Butera. Coding image sequences for interactive retrieval. *ACM: CACM*, 32(7):852–860, july 1989.

[17] Peter Litwinowicz and Lance Williams. Animating images with drawings. In *Computer Graphics, Annual Conference Series, 1994*, pages 409–412. Siggraph, 1994.

[18] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE PAMI*, 11, July 1989.

[19] Steven R. McCanne. *Scalable Video Coding and Transmission for Internet Multicast Video*. PhD thesis, University of California,, Berkeley, December 1996.

[20] D. P. Mitchell. Generating antialiased images at low sampling densities. *Computer Graphics (SIGGRAPH'87)*, 21(4):65–72, July 1987.

[21] J. OŔourke. *Computational Geometry in C*. Cambridge University Press, 1993.

[22] Sassan Pejhan, Mischa Schwartz, and Dimitris Anastassiou. Error control using retransmission schemes in multicast transport protocols for realtime media. *IEEE/ACM Transactions on Networking*, 4(3):413–427, June 1996.

[23] M. Podolsky, C. Romer, and S. Mccanne. Simulation of fec-based error control for packet audio on the internet. *INFOCOM 98*, march 1998.

[24] Thomas Porter and Tom Duff. Compositing digital images. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 253–259, July 1984.

[25] Injong Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *ACM Communication Architectures, Protocols, and Applications (SIGCOMM) 1998*, pages 290–301, Vancouver, B.C., 1998.

[26] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(4):413–424, July 1986.

[27] S Tsekeridou, I Pitas, and C LeBuhan. An error concealment scheme for mpeg-2 coded video sequences. *ISCAS '97*, pages 1289–1292, June 1997.

[28] P. H. Westerink, J. H. Weber, and J. W. Limpers. Adaptive channel error protection of subband encoded images. *IEEE Transactions on Communications*, 41(3):454–459, March 1993.

[29] X. Rex Xu, Andrew C. Myers, Hui Zhang, and Raj Yavatkar. Resilient multicast support for contininuous-media applications. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSDAV) 1997*, pages 183–193, St. Louis, MO, May 1997.

[30] W. Zeng and B Liu. Geometric structure based directional filtering for error concealment in image video transmission. *SPIE vol 2601, Wireless Data Transmission, Photonics East '95*, Oct 1995.