Approximate Storage of Compressed and Encrypted Videos

Djordje Jevdjic
University of Washington

Karin Strauss
Microsoft Research

Luis Ceze
University of Washington

Henrique S. Malvar
Microsoft Research

Abstract

The popularization of video capture devices has created strong storage demand for encoded videos. Approximate storage can ease this demand by enabling denser storage at the expense of occasional errors. Unfortunately, even minor storage errors, such as bit flips, can result in major visual damage in encoded videos. Similarly, video encryption, widely employed for privacy and digital rights management, may create long dependencies between bits that show little or no tolerance to storage errors.

In this paper we propose *VideoApp*, a novel and efficient methodology to compute bit-level reliability requirements for encoded videos by tracking visual and metadata dependencies within encoded bitstreams. We further show how *VideoApp* can be used to trade video quality for storage density in an optimal way. We integrate our methodology into a popular H.264 encoder to partition an encoded video stream into multiple streams that can receive different levels of error correction according to their reliability needs. When applied to a dense and highly error-prone multi-level cell storage substrate, our variable error correction mechanism reduces the error correction overhead by half under the most error-intolerant encoder settings, achieving quality/density points that neither compression nor approximation can achieve alone. Finally, we define the basic invariants needed to support encrypted approximate video storage. We present an analysis of block cipher modes of operation, showing that some are fully compatible with approximation, enabling approximate and secure video storage systems.

CCS Concepts • **Computer systems organization** → **Reliability**; • **Hardware** → **Memory and dense storage**; • **Security and privacy** → *Security requirements*; Digital rights management; • **Computing methodologies** → *Image compression*

ASPLOS '17 April 8–12, 2017, Xi'an, China.

DOI: http://dx.doi.org/10.1145/3037697.3037718

Keywords Approximate Storage; Video Encoding; Multi-Level Cells; Encryption

1. Introduction

Images and videos have been growing in importance and are now prominent consumers of storage in both personal and cloud environments [7]. Trends such as personal action cameras, the use of 360 degrees video, and virtual reality are expected to exacerbate video storage needs even further [19], calling for dense and economical large scale storage of videos.

Approximate storage enables increases in storage density for data items that do not need bit-by-bit precision [5, 17]. For example, single bit flips in raw images affect only a small portion of the image, many times being imperceptible. However, when encoded and stored approximately, these image files may suffer large distortions. This happens because when an image is encoded, each bit carries more information than bits in a raw image, and many pixels depend on that information. Moreover, errors in different bits typically have different impact on the quality of the decoded output. Guo et al. have shown that one particular class of image encoding algorithms can be adapted to partition bits by importance and apply different levels of approximation to each category [5]. The result is higher image storage density with little quality degradation.

Approximate storage of video brings additional challenges. First, in addition to the spatial and encoding dependences already present in encoded images, video also presents temporal dependences. As a result, a single bit carries an order of magnitude more visual information compared to images. To illustrate the problem, a single bit flip can severely damage 5 seconds (100-300 frames) of a video. Second, due to digital rights management (DRM), and for privacy reasons, videos often need to be encrypted. Unfortunately, the common encryption algorithms affect the approximability of encrypted content by creating an additional layer of dependences between bits.

To tackle the above challenges, in this paper we study dynamic data dependences in H.264-coded videos to track the propagation of errors induced by bit flips, based on which we compute the importance of each bit with respect to the visual damage flipping that bit would cause. We add an analysis

framework to the encoder as a post-processing step to assign the optimal approximation level for each class of bits based on the impact they have on the integrity of other data and the overall video quality. Finally, we partition encoded videos into multiple streams, storing each of them with a different level of error correction according to its reliability needs (i.e., approximation tolerance). We further study the security requirements for approximate video storage, analyze multiple modes of a popular encryption algorithm, AES, and show how to encrypt videos while still preserving the ability to approximate them.

To evaluate our approximate storage scheme for videos, we integrate *VideoApp* into a widely used open-source H.264 VideoLan encoder [3]. We show that with a dense but unreliable storage substrate, multi-level cell (MLC) phase-change memory (PCM), it is possible to eliminate 47% of the error correction overhead under the most error-intolerant encoder settings. As a result, we achievie an improvement of 2.57x in density compared to single-level cell (SLC) storage and 12.5% improvement compared to MLC storage with uniformly applied error correction, while only affecting the quality by less than 0.3dB, and while preserving the ability to encrypt the contents.

The rest of the paper is organized as follows: Section 2 provides the background on approximate storage and video encoding. In Section 3, we study the propagation of bit-flip induced errors in H.264-encoded videos. Section 4 presents *VideoApp*, a practical methodology for approximate storage of videos, and Section 5 discusses encryption for approximate video storage. Section 6 describes our evaluation methodology and Section 7 presents the evaluation results. In Section 8 we discuss how relaxing our conservative assumptions can further increase storage gains. Related work is presented in Section 9 and Section 10 concludes the paper.

2. Background

2.1 Approximate Storage

Not all information needs to be stored precisely. Many applications, such as machine learning and signal processing are already noise-tolerant due to inherently noisy inputs. They can also leverage the fact that human senses many times cannot perceive imperfections in the output. For such applications, providing a fully precise storage substrate can be a waste; approximate storage is sufficient [17]. However, certain applications already exploit limitations in human senses to deterministically encode the data and create a smaller storage footprint. Applying approximate storage to their encoded data indiscriminately may create unacceptable outputs.

Prior work has studied how to apply approximate storage to progressively encoded images [5]. Progressive image encoding creates multiple bitstreams, where each subsequent bitstream is used to refine the quality produced by the previous bitstreams in an iterative manner. The main observation was that, in a progressively encoded file, bits that are created in later iterations of the quality refinement process inherently have less impact on the quality of the output compared to the initial bitstream, and consequently, lower reliability requirements. Guo et al. leveraged this observation to partition bits in progressively encoded images into three categories, very important, important, and not important, and assign different levels of error correction to them to achieve higher storage density [5]. The storage substrate used by Guo et al. is a multi-level PCM tuned to minimize error rate for a particular refresh rate. In this paper we assume the same storage substrate.

2.2 Multi-Level Cell (MLC) Memories

MLC memories provide a mechanism to increase density at the device level by storing more than one bit per cell. Resistive memories, such as PCM, are written by applying strong currents to a cell, altering the physical structure of the material and significantly changing its resistance. The cell content is read by running a current/voltage through the cell and sensing the resulting voltage/current. A single-level cell may store one of two values depending on the resistance, whereas multi-level cells are implemented by further dividing the available resistance range into more than two levels, increasing the density commensurately.

Designers working on MLCs have to make difficult choices in trading off density, cost, and reliability of such memories. To make these cells precise (i.e., with extremely low error guarantees, usually 10^{-16}), they need to include more sensitive and advanced circuitry to detect minute differences in resistance, prevent or account for resistance drifts, which is costly. A common alternative is to use cheaper circuitry and apply advanced error correction to the entire storage [18]. The cell design complexity is reduced, but at the cost of additional storage overhead for error correction codes, which can be significant for a dense and error-prone substrate.

Approximation relaxes error rate requirements on the cells and allows designers to tune the reliability of cells to the type of data that they are to store. For example, Guo et al. have used approximation to tune a MLC PCM substrate for images [5]. Two types of errors affect the accuracy of PCM: write/read errors, related to the access circuitry, and drift-induced errors that reflect non-linear drifts in resistance over time. To minimize overall cell error rates, Guo et al. first apply non-uniform partitioning of the cell resistance range into multiple levels to account for non-linear resistance drifts, and then equalize write/read error rates with drift error rates for a particular scrubbing frequency (three months by default). Then, they modify a variant of the JPEG-XR encoder to partition data with different levels of importance into three categories. Finally, they study what level of approximation (i.e., error rates) each category can tolerate without allowing the overall quality to degrade by more than 1dB. The result is a very dense MLC PCM substrate that selectively uses

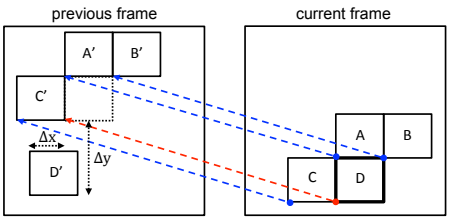

Figure 1. Example of predictive metadata coding of motion vectors. The blue dashed lines correspond to the motion vectors for MBs A, B, and C. The motion vector for MB D is predicted based on the vectors for A, B, and C, and its predicted value is shown in red. Only Δx and Δy, which compensate for any misprediction, will be encoded.

error correction only where needed, improving density and resulting costs.

2.3 Video Encoding

Video encoding includes a variety of lossless and lossy video compression techniques. In this paper, we study lossy video encoding techniques due to their widespread use and their ability to achieve far better compression ratios with tolerable quality loss. In particular, we focus on one of the most commonly used formats, H.264/MPEG-4 AVC [2], widely used both for video storage and streaming [15]. Our methodology, however, applies to most modern video encoders.

2.3.1 Frames and Macroblocks

H.264 encodes videos frame by frame, creating three types of coded frames. I-frames are self-contained; they depend only on the previously encoded content within the same frame, similar to images. Being limited to spatial prediction (*intra prediction*), I-frames have the lowest compression ratios, and encoders periodically insert them as checkpoints to refresh the stream and limit the propagation of eventual errors, at the expense of extra storage. Between I-frames encoders create many P- and B-frames, whose content is predicted based on their similarity to other frames. Frames are further divided into 16x16 pixel groups called macroblocks (MB), which are the basic encoding units. An encoded frame consists of a header, which holds information that applies to the entire frame, and its MBs, which are encoded one by one in the raster scan order—i.e., row by row, from left to right, starting from the top let corner of the frame.

2.3.2 Encoding stages

H.264 encoding features two major and mutually dependent stages [2, 15]. During the first stage, the encoder seeks to find high visual redundancy between spatially or temporally correlated groups of pixels, and computes pixel-by-pixel differences between them (*pixel-level prediction and compensation*). It leverages the fact that neighboring pixels have highly similar values, both within a frame and across consecutive frames. For example, the pixel values for macroblocks A, B, and C in Figure 1 can be better compressed in the next stage if represented as a difference to the most similar respective regions of the same size in a previous frame, together with the motion vectors (dashed blue lines) that identify those regions in the previous frame. The second stage of the process (*coding*) seeks to compactly encode the values determined by the first stage and produces the coded bitstream. If the resulting frame size violates quality/compression rate targets, the encoder will iteratively seek to find better solutions by looking for alternative visual redundancies in the first stage that could lead to more compact coded bitstreams in the second stage.

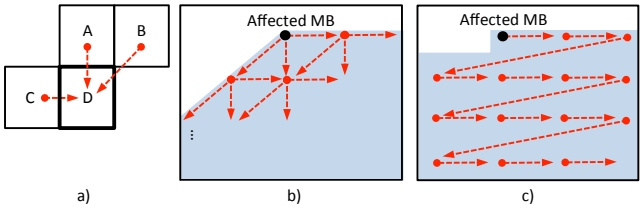

Figure 2. a) Coding dependency pattern between MBs, b) Error propagation and the affected frame area in the ideal case, and c) Actual error propagation pattern within a frame.

The coding stage involves several tasks. The encoder first transforms the computed pixel differences into a frequency domain using the Discrete Cosine Transform, achieving a significant compression ratio at the expense of negligible quality losses (*transformation*). It further performs controlled quantization of the resulting transform coefficients to store them more compactly (*quantization*). Quantization provides the main quality/storage control knob for video encoding. To encode the video metadata, such as motion vectors or quantization levels, the encoder first performs static spatial prediction to find the expected metadata values, based on the corresponding values in surrounding MBs, and then encodes only the difference between the actual values and the expected ones (*predictive metadata coding*). For example, neighboring parts of a moving object will likely move at similar speeds, and the movement of one of them can be accurately predicted based on the movement of the others. In Figure 1, the motion vector for currently encoded MB D is first predicted as a median of the motion vectors corresponding to already encoded surrounding MBs A, B, and C, and only the difference between the prediction and actual motion is encoded.[1] The final task includes *entropy coding*, which losslessly encodes already compressed data in a way that leverages any remaining predictability, producing the final bitstream.

2.3.3 Macroblock structure

A coded MB consists of several *syntax fields* that describe its content. A MB header defines the type of prediction

[1] Macroblocks A, B, and C are always encoded before macroblock D due to the raster scan order.

used to encode it, which can be spatial (intra-frame) or temporal (inter-frame). In case of intra prediction, the header encodes the spatial direction of the prediction, defining the set of pixels in already encoded surrounding MBs that will be used as a reference for prediction. In case of temporal prediction, the MB header defines the list of frames that the currently encoded MB will reference (one for P-frames, two for B-frames), along with the *motion vectors* that define the referred set of pixels within the referenced frames. After the header, a MB contains coded quantization parameters, which define the level of quantization applied to transform coefficients. Finally, the rest of the bits describe the layout and values of the quantized transform coefficients.

2.3.4 Entropy Coding

H.264 uses two entropy coding algorithms to produce the final bitstream: Context-Adaptive Variable-Length Coding (CAVLC), and Context-Adaptive Binary Arithmetic Coding (CABAC). On the one hand, CAVLC-encoded videos require considerably less compute power to encode/decode and are more error-tolerant compared to CABAC-encoded ones [15]. On the other hand, CABAC-encoded videos provide up to 15% better compression [11], while being highly computationally intensive and showing high sensitivity to even smallest errors such as bit flips. Although CAVLC would be a better match for approximate storage, in this paper we choose to study CABAC, for two main reasons. First, we wanted to be highly conservative and choose the most storage-efficient scheme despite its error intolerance. Second, CABAC is used in all recent and currently developing standards, such as H.265/HEVC, Google's VP9, and Mozilla's Daala [1], due to its unparalleled compression efficacy, and it is likely to be the most prevalent entropy coder in the future.

One of the fundamental advantages of CABAC is its use of arithmetic coding, which allows coding events to be assigned a non-integer code length [11], making it possible to represent a symbol of high frequency with less than one bit. Each coded sequence can be assigned a unique range of rational numbers between 0 and 1, where the width of the range corresponds to the probability of that sequence occurring in the current context. CABAC represents each sequence with the most compact rational number that falls in the specified range, and in doing so, it makes the length of each coded sequence inversely proportional to its occurring probability, approaching the theoretical compression limit established by Shannon's source coding theorem [10]. CABAC computes the probability of each sequence by modeling a statistical *context* for each field of each MB in the current frame. To model the context of a currently encoded/decoded MB, CABAC relies on the entropy coder state in surrounding MBs, as shown in Figure 2(a), respecting the scan order.

3. Propagation of Bit-Flip Induced Errors

A single bit flip in an H.264 video can cause large visual distortions in many ways. At the lowest level (entropy coder), sequences are encoded as rational numbers. Upon a bit flip, the entropy decoder will, in the ideal case, intepret the affected sequence as a different rational number, hopefully similar to the original one. However, the decoded sequence is not necessarily semantically similar to the original one [11], which can cause misinterpretations of arbitrary magnitude. Moreover, because CABAC is a context-aware coder, the bit flip will wrongly update the statistical context, changing the probabilities of zeros and ones that both the decoder and encoder must consistently maintain for each syntax field. Changes in the context cause misinterpretation of the same field in the rest of the frame, following the spatial pattern shown in Figure 2(b).

The visual manifestation of errors depends on the affected syntax field. In case the flipped bit corresponds to transform coefficients, the inverse transformation will amplify the error, causing unpredictable visual damage within the MB. An error in quantization parameters will similarly lead to wrong interpretation of the transform coefficients. An error in motion vectors will result in referencing a wrong group of pixels or a wrong frame. Furthermore, the predictive coding of metadata fields creates dependencies (Figure 2(a)) that will propagate any change in the interpretation of metadata, following the pattern shown in Figure 2(b). For instance, to compute the quantization level for MB D, the decoder first computes the median of quantization levels applied to already decoded MBs A, B, and C, forming a prediction of the quantization level for MB D. It then interprets the coded quantization level as a difference it needs to add to the predicted value. A quantization error in any of the surrounding MBs (A, B, or C) will affect MB D and propagate further. Notice that the error propagation due to predictive coding happens on top of the entropy-coding related propagation.

The above analysis assumes that the beginning and the end of each syntax element in the damaged bitstream can be properly identified, in which case the affected frame area is the one shown in Figure 2(b). Unfortunately, the distortions that the entropy decoder accumulates eventually will leave it out of sync with the encoder, breaking the structure of the stream and leading to misinterpretation of the subsequent syntax elements in the stream, and consequently, to intractable visual damage that affects the entire area shown in Figure 2(c). Fortunately, the entropy context is reset at the beginning of every frame, allowing the encoder and decoder to resynchronize.

Finally, all the visual damage accumulated within the affected frame will be propagated in both time and space to all the frames that reference the damaged regions. Such propagation is a consequence of the first encoding step, which encodes each MB as a difference (pixel-by-pixel) relative to the most similar previously encoded MB. Unlike the cod-

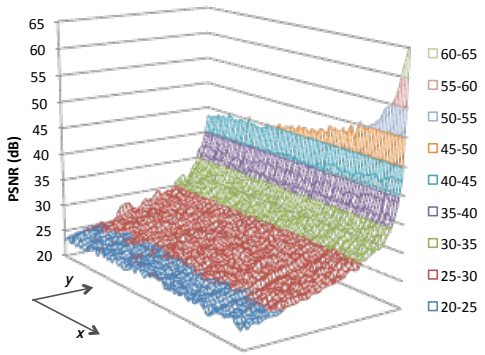

Figure 3. Frame PSNR after a single bit flip as a function of the affected MB position within the frame. The origin corresponds to the top left corner of the frame.

ing stage, which shows a static error propagation pattern, the compensation stage propagates errors in the pixel domain and in a data-dependent way.

We refer to the errors accumulated by the coding stage as *coding errors*, and the errors that are the result of the propagation in the pixel domain as *compensation errors*. Coding errors are responsible for most of the damage observed in the frame containing the flipped bit, but their reach is limited by the frame boundaries. Compensation errors propagate the visual damage caused by coding errors to other frames by referencing damaged regions of the source frame, adding wrong pixel values to correctly decoded pixel differences. Unlike coding errors, compensation errors will not affect the interpretation of metadata fields in the affected frames.

3.1 Effects of Coding Errors

As a consequence of the coding error propagation pattern shown in Figure 2(c), the suffered quality loss highly depends on the spatial location of the affected MB within its frame. To test this hypothesis, we perform an experiment in which we inject one bit flip at a time, controlling the location of the affected MB within the frame, and measure the resulting quality loss. The results are averaged over hundreds of frames per MB position. Figure 3 shows the impact of the location of the flipped bit on quality. The x and y axes correspond to the coordinates of the affected MB, with point (0, 0) corresponding to the top left corner of the frame. The quality is measured as peak-signal to noise ratio (PSNR) relative to the coded video without bit flips, with higher values corresponding to higher quality. We analyze only frames that use no intra-prediction to exclude the effects of compensation errors. We conclude that the coding error propagation indeed matches the pattern shown in Figure 2(c), with the bit flips occurring in MBs spatially closer to the bottom-right corner causing much less damage compared to MBs closer to the top left corner (point (0, 0)).

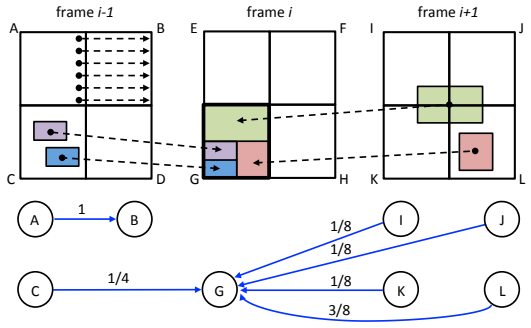

Figure 4. Modeling temporal and spatial dependencies.

4. VideoApp

We present *VideoApp*, a framework that accepts an encoded video as input and orders all its bits with respect to the visual damage they would cause if flipped. Because the precise damage a given bit flip will cause is not possible to determine in a practical way, we use a heuristic that estimates the number of macroblocks each bit flip would damage, which we call $importance$. Importance is by definition proportional to the damaged surface area, which correlates well with the observed quality loss for all major metrics. We provide an algorithm that accurately estimates the number of damaged MBs by tracking dependencies between MBs. $VideoApp$ creates a weighted dependency graph, in which the nodes are MBs, and the edges represent the dependencies between them. The weight of each edge is between 0 and 1, and it is proportional to the visual damage that an error in the source MB would transfer to the destination MB. Having constructed the entire graph, the tool traverses it backwards to compute the importance of bits in each MB.

4.1 Modeling Compensation Error Propagation

Compensation errors propagate in the pixel domain by referencing damaged regions, which do not need to be aligned to MB boundaries. A MB can therefore depend on several other MBs by referencing a single 16x16 group of pixels. Furthermore, to allow for higher precision, H.264 can use smaller MB partitions as basic compensation units. These units can have different sizes, which we model in detail: 16x8, 8x16, 8x8, 4x8, 8x4, and 4x4. Our model assumes that the visual damage in source MB *X* will propagage to destination MB *Y* to the extent that is proportional to the number of pixels in MB *X* that are referenced by MB *Y*.

The example in Figure 4 illustrates how *VideoApp* models temporal and spatial compensation dependencies. For illustration purposes only, the three frames at the top have each four MBs. MB G in frame i is divided into four partitions, and each of them is compensated independently. The weighted graph at the bottom represents the dependencies among the depicted frames, with weighted edges representing the relative area compensated in MB G. Multiple dependencies from one MB to another can be aggregated into a

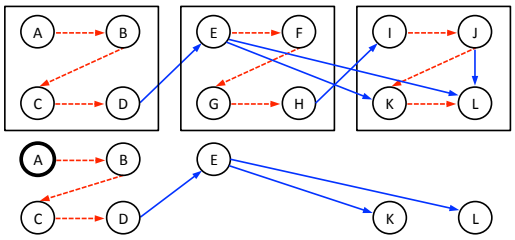

Figure 5. Error propagation tree for a bit flip in MB A.

single edge by adding up their weights. This is the case for edges C→G and L→G in our example. For every MB the sum of weights of its incoming edges (if any) is always 1.

MB B in frame i-1 is an example of intra-frame prediction, in this case in the horizontal direction. The entire MB predicted by extrapolating the last column of 16 pixels in MB A, and the per-pixel difference is encoded for compensation. For certain prediction directions, the set of extrapolated pixels may belong to multiple MBs, in which case we create a weighted edge between each source MB and the predicted MB, and distribute the weight of 1 across all MBs proportionally to the number of pixels they contribute.

4.2 Modeling Coding Error Propagation

Coding errors propagate in a static fashion as shown in Figure 2(c) with all coding edges in one frame forming a weighted linked list. Because we define importance as the number of MBs affected by a bit flip, each weight equals 1. By tweaking the coding weights one can give more (or less) importance to coding dependencies compared to compensation dependencies and obtain a more precise estimate of the visual quality loss as opposed to our heuristics that is based only on the damaged surface.

4.3 Computing Macroblock Importance

Figure 5 (top) illustrates the error propagation process within and across frames for a single bit flip that occurs in MB A in the first frame. Upon a bit flip, coding dependencies will create visual damage in MBs B, C and D, according to the static coding propagation pattern. This damage will further propagate through compensation dependencies to other frames, namely to MBs E, J, and K. However, in those other frames, the damage propagated through compensation dependencies will not cause coding errors, because all the metadata there can be properly decoded. In other words, propagation dependencies can be appended to coding dependencies, but coding dependencies cannot be appended to propagation dependencies.

The bottom part of Figure 5 shows the error propagation tree that corresponds to a bit flip in MB A. Knowing the error propagation tree for a MB, one can easily compute its importance. The basic idea is to initialize all the nodes of the tree to 1, indicating that only one MB worth of area would be damaged by a bit flip, which is true only for the leaves of the propagation tree. Starting from the leaves, we traverse the propagation tree and update the importance of each MB by adding the weighted sum of its children, where the weights correspond to the damaged area transferred to the children. At the end of this process, the importance of MB A will reflect the total area (in MBs) to which an error originating in MB A will propagate. The following algorithm leverages this observation to compute the importance of each MB given the entire dependency graph produced by *VideoApp*:

1. Create a graph containing only compensation dependencies;
2. Initialize the importance of each node (MB) to 1;
3. Sort the graph topologically;
4. Starting from the end of the sorted list, update the importance of each node by adding the weighted sum of importances of all children, where the weights are the compensation dependency weights computed by *VideoApp*. By the end of this step the importance of each MB will reflect the number of MBs to which an error in a given MB will propagate through compensation;
5. Create a graph containing only coding dependencies;
6. Initialize the importance of each node (MB) to the importance computed in step #4;
7. Sort the graph topologically;
8. Starting from the end of the sorted list, update the importance of each node by adding the weighted sum of importances of all children, where the weights are the coding dependency weights.

4.3.1 Time and Space Overheads

The proposed methodology is meant to be used only once per video, during or after encoding. Our implementation incurs a 2-3% time overhead relative to encoding, with the most complex operation being topological sort. The temporary graph structures can be larger than the corresponding video when encoded in standard quality, yet they are an order of magnitude smaller than the raw video being encoded. Moreover, for large videos, both the time and space overhead can be significantly reduced because the importances can be independently computed for short sequences between I-frames, thanks to the streaming nature of videos. Namely, steps 1–4 do not need to be performed on the entire graph at once, but instead they can be independently performed on each connected component between two I-frames. Similarly, steps 5–8 can be performed independently for each frame. This allows for efficient and even real-time implementations of our methodology.

4.4 Assigning Error Correction

Each coded frame consists of a frame header and all MBs encoded in the scan order. Because corrupting the frame header would destroy the entire frame, we assign it the strongest er-

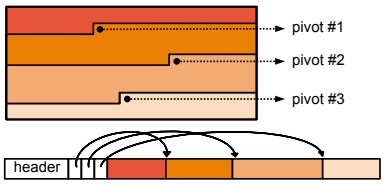

Figure 6. Dividing a frame into reliability areas through pivots, respecting the coding error propagation pattern.

ror correction scheme, which corresponds to precise storage (10^{-16}). To every MB in the frame we assign an error correction scheme that matches its importance. We establish the relationship between the importance and error correction requirements empirically across a wide range of videos (Section 7). For each scheme, we find the importance threshold that a MB needs to reach to be protected by that scheme, in a way that minimizes the quality loss and maximizes the storage benefits. However, the flexibility of fine-grained error correction assignment implies a high bookkeeping overhead because it requires keeping track of the correction levels assigned to each individual MB, the overhead of which can be large as the video itself. Instead, we make the observation that the coding error propagation pattern in Figure 2(c) imposes a strictly decreasing order of MBs within a frame with respect to the importance computed by our algorithm. In other words, a MB will always have higher importance than the one that succeeds it in the scan order within the same frame.

The fact that within a frame MBs are sorted by importance allows us to precisely and compactly describe the importance layout of a frame. Given the importance of each MB in a frame, and given the importance threshold for each error correction scheme, we can describe the error correction level of all MBs in a frame with only a few pivot points that signal a change in the error correction scheme. Figure 6 illustrates the error correction assignment using pivots. Leveraging the importance order of MBs within a frame allows us to reduce the bookkeeping storage overhead to only a few bytes per frame. The pivot locations and their importance threholds are added to the frame header and stored precisely with the strongest error correction scheme (10^{-16}).

5. Encrypting Approximate Video Storage

Digital Rights Management (DRM), commonly associated to videos, usually employs encryption to protect the contents. Besides protecting privacy, encryption algorithms often protect data integrity against tampering, either as a feature or as a side effect of their efforts to ensure privacy. Unfortunately, this also means that for most of today's storage encryption schemes corrupting a single bit in an encrypted file makes the entire file unintelligible. This makes approximate storage of such encrypted videos infeasible because approximation intentionally allows single bit corruptions. In this section, we define the requirements for encryption on top of approximate storage and show how an appropriate encryption scheme can be constructed.

5.1 Encryption Requirements

To allow for approximate storage, an encryption scheme should:

1. Make the content unreadable to non-authorized parties;
2. Not become completely unreadable due to individual bit flips, i.e., errors in encrypted bits should not propagate throughout the rest of the video;
3. Not interfere with the ability to approximate data, i.e., approximating an encrypted file should produce output of the same quality as an already approximated file.

5.2 Finding Compatible Encryption Modes

Storage systems typically use block ciphers to encrypt data, AES encryption being one of the most popular [14]. Its basic operation is a substitution-permutation network (subperm in Figure 7). The network has two inputs, a 16-byte input data block to encrypt, and the secret encryption key. The key defines the secret transformation performed by the network on the input data to produce a 16-byte encrypted output.

Algorithms implementing block ciphers can be used in a variety of modes [14]. Figure 7 shows four modes of operation. They differ in whether they use only a substitution-permutation network or an additional exclusive OR (XOR) step, how these steps are arranged, and whether the computation of blocks depends on computation from previous blocks.

Electronic Codebook (ECB) is the simplest of them. It takes plaintext and the key as inputs and returns ciphertext. Despite making each of the blocks unreadable, it is vulnerable to dictionary attacks over the collection of blocks, since a particular 16-byte value will always be encrypted to the same target value. It thus fails requirement #1 above. To address this problem, Cipher Block Chaining (CBC) uses the ciphertext from the previous block to further randomize the current value using XOR. The effect is that a single plaintext value can result in a large number of ciphertext values, making the blocks collectively unreadable and finally meeting requirement #1. However, this mode fails requirements #2 and #3 because now when an error occurs in ciphertext, it propagates to all subsequent blocks.

Output Feedback (OFB) and Counter-Mode (CTR) meet all requirements. OFB differs from CBC in that, instead of creating a dependence through the previous block's ciphertext, it creates a dependence through an initial seed value processed by the substitution-permutation network multiple times, more specifically once per block (previous subperm'd value). While making the video unreadable, its dependence propagation is done independently of the value that is stored as ciphertext. In other words, as long as the key and the initial value are intact, errors in ciphertext storage do not propagate to later blocks, i.e., the error affects only the flipped

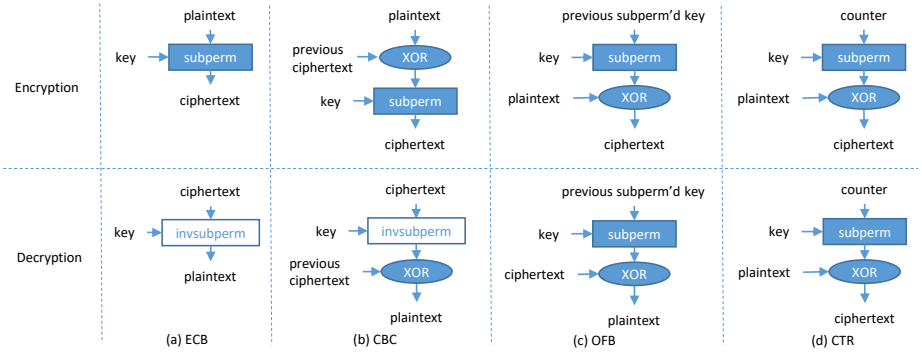

Figure 7. Various block encryption/decryption modes: (a) Electronic Codebook (ECB); (b) Cipher Block Chaining (CBC); (c) Output Feedback (OFB); and (d) Counter-mode (CTR). Boxes named $subperm$ represent a substitution-permutation network and boxes named $invsubperm$ represent their inverse function.

bit. Finally, CTR offers the same properties, this time propagating the dependence implicitly via a counter that is incremented for every block.

Generalizing, modes appropriate for use with approximate video storage need to randomize the mapping between plaintext and ciphertext values while not creating chains of values that are kept in approximate storage (i.e., ciphertext).

5.3 Encrypting Multiple Streams

Approximate video storage splits data into multiple streams during encoding and creates the metadata needed to merge the streams in the header of each frame, which is kept in precise storage. After encoding, each of these streams is encrypted separately. If an initial value is required to seed the encryption, it is derived from a single value for all streams pre-appended to each stream's identifier. It is important to note that our approximation methodology must be applied before encryption, and therefore it must be trusted.

6. Evaluation Methodology

We use VideoApp for two purposes: first, to profile a set of videos at a wide range of quality targets and establish the target approximation levels, and second, to separate the original bitstream into multiple bitstreams according to the selected approximation levels. For evaluation purposes, we integrate VideoApp with a widely used open-source VideoLan H.264 video encoder [3]. Our methodology, however, works with any H.264 encoder.

6.1 Metrics

We evaluate the proposed scheme through a combination of two metrics: peak-signal to noise ratio (PSNR), averaged across frames, and storage density. Average PSNR is commonly used in the video processing community to measure the quality of reconstructed videos. It compares the original video—frame by frame, pixel by pixel—against the decoded video that contains errors from lossy compression (e.g., quantization) and the storage system (in our case, uncorrected read/write and drift errors). The higher the PSNR, the smaller the difference between the original and the reconstructed video. While we present the results only for PSNR, we also studied other metrics supported by our quality measurement tool [13], which include Structural Similarity (SSIM), Multi-Scale Structural Similarity (MS-SSIM), and Visual Information Fidelity (VIFP). For all metrics we measure the average value across the frames, following the established practice. We found that our methodology relates well to all of these metrics in case of bit-flip related distortions, but we only report average PSNR for space reasons. Although SSIM may produce better correlation with subjective results [21], the improvement is not significant, especially in the context of approximate storage. A recently announced subjective metrics from Netflix [20] shows promising results, but it is specifically tailored to exclude quality assessment in case of errors in the bitstream and is therefore not suitable for our study. In contrast, PSNR is well understood and can be more meaningful in applications where the consumer of the decoded video can be an analytics or machine learning system, rather than a human observer.

The standard storage density metric in the multimedia community is the number of pixels per bit. Because we use multi-level cells, we define the storage density as the average number of pixels that can be stored in a single cell.

6.2 Storage substrate

We use a highly optimized multi-level cell PCM substrate [5] that minimizes the overall error rate by biasing the level ranges so as to equalize the error rate at each level with the resistance drifts accumulated over the scrubbing (refresh) interval of 3 months. The resulting substrate achieves an error rate of 10^{-3} with 8 levels per cell, providing a 3x higher density at the expense of frequent errors. To provide different reliability levels, we add a variable amount of ECC to different bits, but store them all in the same substrate to minimize the cost. We use advanced BCH-X error correction codes, where X denotes the number of errors the code can

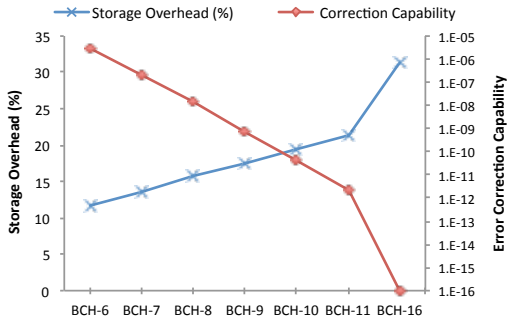

Figure 8. Error correction overhead (left) and capability (right) for 512-bit blocks with a raw bit error rate of 10^{-3}.

correct within the storage block it protects, which include both the data block and the code metadata (the codes are self-correcting). Figure 8 shows the codes used, their storage overhead, and their error correction capability given a storage substrate with raw bit error rate of 10^{-3}. For example, BCH-6 can correct up to 6 errors in a 512-bit PCM block (plus 60 bits of metadata), and for a raw bit error rate of 10^{-3} it produces the resulting uncorrectable error rate of around 10^{-6} with an 11.7% storage overhead.

6.3 Input videos and Encoding Parameters

Our quality measurements are done on a collection of 14 raw video sequences distributed by Xiph.Org Foundation [22]. The sequences are 500-600 frames long, capturing 50-60 frames per second, and their resultion is 1280x720. Setting encoding parameters in H.264 video encoding is a complex task: unlike image quality, video quality is rarely controlled by setting a target PSNR or quantization because the encoder will encode fast moving objects more aggressively to leverage the fact that human eyes cannot perceive high levels of details in them. We therefore control the video quality using the constant rate factor (CRF) [12], following the common practice in the video compression community, and evaluate our scheme at three quality targets: standard quality (CRF=24), high quality (CRF=20), and very high quality (CRF=16). Setting quality via CRF has a high correlation with both PSNR and subjective visual quality [12].

6.4 Simulation

We conduct Monte Carlo simulations to model read- and write-induced errors caused by circuit imprecision. Due to the nondeterministic error patterns, we run each video in the suite through our stochastic model 30 times, with errors appearing at random locations every time. For low error rates (3-4 errors per video), we make sure that the number of errors per video follow the binomial distribution across 30 runs. For very low error rates (e.g., 10^{-12}), we make sure that at least one bit flip happens in each video, and then scale down the quality loss by multiplying it by the probability that the error happens within the video of a given size.

Table 1. Error Correction Assignment.

Importance	Scheme	Error rate	Overhead
0-2	None	10^{-3}	-
3-10	BCH-6	10^{-6}	11.7%
11-13	BCH-7	10^{-7}	13.65%
14-16	BCH-8	10^{-8}	15.6%
17-20	BCH-9	10^{-9}	17.55%
21-26	BCH-10	10^{-10}	19.5%
frame header	BCH-16	10^{-16}	31.3%

We decode approximate videos using the *ffmpeg* library, and evaluate the quality using the VQMT video quality measurement tool [13]. Importantly, we report the *maximum* (rather than the average) quality loss for each video, which gives us a highly conservative estimate of the quality loss.

7. Evaluation

7.1 Methodology Validation

Given a H.264-coded input video, *VideoApp* outputs per-MB importance corresponding to the number of blocks a bit flip in that macroblock would affect. For the set of videos we consider, the estimated importance of MBs largely varies, ranging from 1 to 2^{26}. To validate the tool and to test our hypothesis that the definition of importance above relates to the visual quality loss, we need to verify that flipping bits in MBs of lower importance, as computed by the tool, will cause less damage compared to bit flips occurring in MBs of higher importance. To do so in a practical way, we set up the following experiment: we sort all MBs in a video by importance, and then divide them into 16 bins equal in storage so that the first bin (bin 0) contains the least important MBs (as computed by the tool) that make 1/16h of the total bits in the video. Every subsequent bin contains the next set of least important macroblocks. The last bin, bin 15, contains the 1/16th bits of highest importance. We inject errors in one bin at a time, while keeping the other bins precise so that we can directly compare the quality loss from bins of different importance. The bins are equally sized to ensure that differences in quality do not come from differences in the number of bit flips in each bin.

Figure 9(a) shows the visual quality degradation for different bins (labelled 0 to 15) as we vary the error rates on the x-axis, whereas Figure 9(b) shows the maximum MB importance in each bin on a logarithmic scale (base 2). The similarity between bins (right) is correlated with the gaps between quality degradation curves (left). Independent of the gap size, the order of the quality degradation curves on the left strictly follows the bin importance order on the right. We conclude that the MB importance levels computed by *VideoApp* correlate well with objectively measured quality degradation.

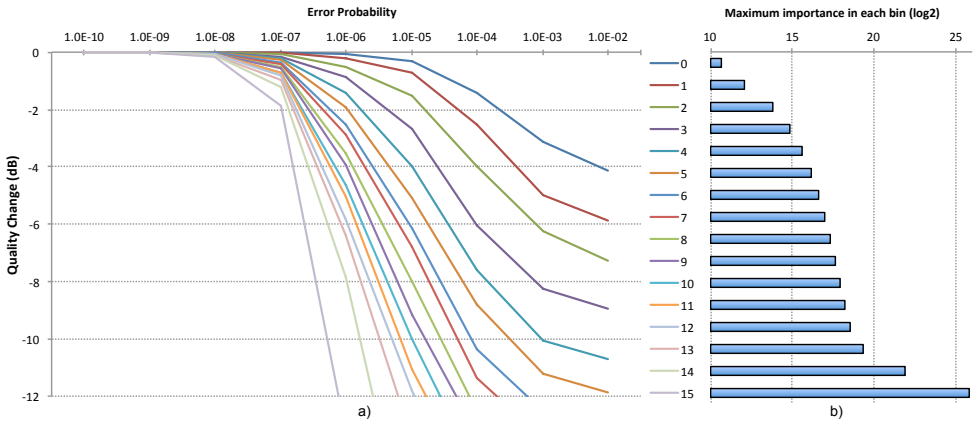

Figure 9. a) Quality loss in dB as a function of the error rate across bins of the same size but different importance, b) maximum importance in each bin on a log2 scale. The bins are enumerated in the order of importance, from lower to higher.

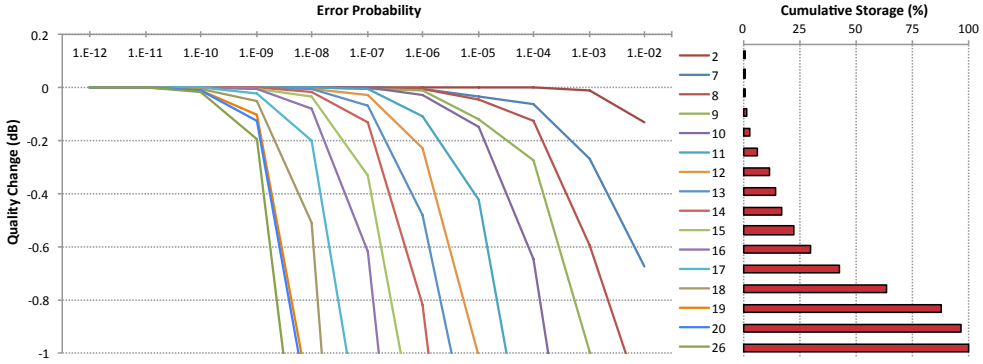

Figure 10. a) Cumulative quality loss as a function of the error rate for different importance classes. The number next to each curve corresponds to its importance class, where importance class i contains all macroblocks whose importance is equal or less than 2^i, b) Cumulative storage per importance class.

7.2 Assigning Error Correction To Importance Classes

Our next goal is to relate the MB importance to the required error correction scheme, while minimizing the quality loss and maximizing the overall storage savings. To do so, we need to consider the importance of MBs, the storage they occupy, the quality loss of approximation at a given error rate, the overhead of error correction needed to guarantee the given error rate, and the target quality loss budget.

We first size our quality loss budget so that the worst case quality loss due to approximation is always less than the quality loss we would get with deterministic compression (encoding) for the same storage savings. Given that our storage savings are on the order of 10-15% depending on the error correction assignment, and given that for the same storage reduction the encoder can deterministically compress the videos and lose between 0.4 and 0.6dB in quality, we set the quality loss limit to 0.3dB to guarantee that approximation will always reduce the quality by less than compression would do when judged by an objective quality metric.

To decide how to optimally distribute our quality loss budget, we set up an experiment in which we classify the MBs into importance classes on a logarithmic scale, so that class i contains all MBs whose importance is no greater than 2^i. Figure 10(a) shows the cumulative quality loss (in the quality region of interest) as a function of the error rate for different MB importance classes. For example, the first class contains MBs of importance less or equal than 2. Figure 10(b) shows the fraction of storage that each importance class occupies (also cumulative). To keep the figure readable, we omit a few importance classes that occupy a negligible amount of storage.

The information in Figure 10 and Figure 8 is sufficient to decide on the right correction scheme for each importance class given the quality loss budget. Because the graph on the left shows cumulative quality loss, we can easily compute how much quality we would lose if we choose to assign one error correction scheme to class i and a weaker one to class $i-1$ by doing a simple subtraction. We first distribute our 0.3dB budget to importance classes proportionally to the amount of storage they occupy to maximize the storage sav-

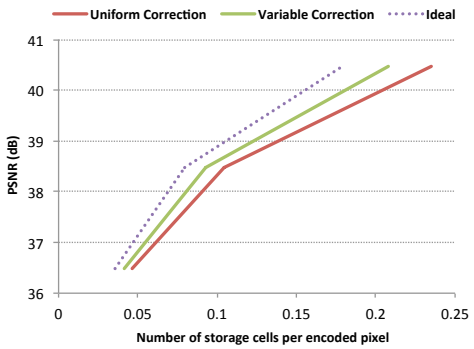

Figure 11. Overall storage benefits of approximation assuming a storage scrubbing (refresh) interval of 3 months.

ings on error correction. We then start with the lowest importance class, and choose the weakest error correction scheme that would cause a quality degradation that is in within the limit assigned to it. Once class i is assigned its correction scheme, we move on to class $i+1$, and assign it the weakest correction scheme that will keep the quality loss within its assigned limit, where the quality loss excludes the bits covered by the previous class. At the end of this algorithm, the quality loss will be distributed across importance classes in a way that chooses the weakest error correction scheme for each class and guarantees the maximum storage savings. The error correction schemes we apply to importance classes based on the above algorithm are listed in Table 1. If needed, the number of error correction schemes can be easily reduced by upgrading the correction scheme that protects the fewest bits to a stronger one. Importantly, the metadata belonging to frame headers is kept precise, by assigning them the strongest error correction scheme (BCH-16, 10^{-16} error rate with 31.3% overhead). Fortunately, these bits amount to less than 0.1% of the total storage.

7.2.1 Alternative Approximation Strategies

Instead of maximizing the storage gains under a pre-allocated quality loss budget, we could assume an alternative correction scheme assignment strategy. Namely, whenever we need to decide whether to protect a class of bits with a stronger scheme or with a weaker one, we could check whether the storage gains under the weaker scheme justify the quality loss when compared to compression. In other words, we go for approximation only when it shows a clear win against compression, otherwise we employ further compression. Such a conservative strategy could result in slightly better quality/density points.

7.3 Overall Storage Gains

Figure 11 shows the storage density expressed as the number of cells per image pixel for three designs. All designs assume an optimized PCM substrate with 8 levels per cell, with the levels optimally biased to minimize the overall error rate [5]. The first design uses uniformly applied error correction to the entire storage, guaranteeing an error rate of at most 10^{-16}. The second design applies variable error correction based on importance computed by *VideoApp*. The last design assumes a perfect error correction scheme that correct all errors with no storage overhead. Overall, by applying error correction in an optimal way we were able to eliminate 47% of the error correction overhead and save 12.5% in storage with strict quality loss guarantees. Compared to a highly reliable single-level cell design with no error correction, we provide 2.57x higher density.

Interestingly, we find that increasing the quality of a video slightly reduces our ability to approximate it. Our intuition was exactly the opposite: larger videos carry less information per bit and should be more tolerant of bit flips. However, a higher quality video contains larger frames, which increases the probability of error in each frame for a fixed error rate, affecting more frames and causing more damage for CABAC-coded videos. Our error correction assignment for the highest video quality is therefore slightly more conservative and results in storage savings of 12.3%. We believe that the observed quality/approximability relationship may be different for other entropy coders.

8. Discussion

H.264 provides the option to limit the coding error propagation in videos by dividing each frame into several slices. Each slice uses a separate entropy context and does not rely on other slices for predictive coding, limiting the propagation of coding errors to the slice boundaries at the expense of extra storage. We deliberately decided not to use more than one slice per frame to remain highly conservative and to be able to establish the lower bound for the benefits of approximation in the context of video storage.

Similarly, using the other entropy coder available in H.264, CAVLC, would significantly reduce the effects of errors. In fact, CAVLC is recommended for applications with frequent errors [15]. However, it does so at a significant storage price (10-15% [11]). We decided to use CABAC, despite its known error intolerance, to remain conservative, and to be able to give a definitive answer to the question "Can approximation bring higher objectively measured benefits compared to deterministic video compression?" The answer is yes. However, moving beyond this question, a more appropriate use of *VideoApp* would likely be in combination with CAVLC, in which case one could achieve the storage efficiency of CABAC (given the benefits of approximation), and low computational complexity and high error resilience of CAVLC. In combination with slicing, the overall storage gains would likely approach the ideal curve in Figure 11.

Finally, we would like to raise the following question: is it possible to design the first stage of video compression (prediction and compensation) in a way that increases the opportunity to approximate videos? We pick this stage, because it is the one that gives the freedom to encoders to

compete, whereas the other components are set in stone by the standard. For example, H.264 provides a flag to disallow B-frames to be used as references, effectively creating many unreferenced frames in which errors cannot propagate to other frames. The amount of unreferenced storage can be further increased by biasing the encoder to create more B-frames, and thus, even more unreferenced frames. After experimenting with such options, we found that many of the quality loss curves in Figure 9 and Figure 10 would significantly shift to the right, tolerating higher error rates. The resulting videos were highly polarized into important and less important bits, which is ideal for approximation. At the same time, as expected, these encoding options often produce suboptimal solutions and increase the storage, sometimes cancelling out the benefits of the increased approximability, but other times resulting in a better overall solution, leaving us without a clear conclusion. Our question to the video compression community is: what would the encoding process look like if it were given an extra metric to optimize for, which is to produce a more approximable video?

9. Related Work

This work is inspired by approximate storage [17] and approximate image storage [5]. We use the same technological assumptions about the approximate storage hardware, but we focus on videos instead of raw data or just images. In their approximate image store study, Guo et al. were able to achieve bigger storage gains [5]. Their study focused on a progressive image format that already separates bits into importance classes, with each subsequent class being used only to refine the image. The format they use limits the error propagation by design not only to the image boundaries, but also to a smaller region within the image. Unfortunately, videos are much more sensitive to errors because the errors propagate through the entire frame (see Figure 3) and well beyond the frame in which they occur. Note that their work is orthogonal to ours: videos could be also encoded in a "layered" way, where each layer refines the quality produced by the previous (scalable video coding). Our work focuses on approximation within a layer, and is trivially extensible to multiple layers by adding another dimension of approximation. In contrast, the work of Guo et al. [5] primarily focuses on differences in reliability across layers.

Guo et al. [6] and Frescura et al. [4] used variable error protection to protect images and videos against transmission channel errors. They use variable coding and redundancy depending on which part of the bitstream is being encoded. Headers are protected using maximum redundancy and less important data is protected using minimal redundancy. Their work differs from ours in that the channel here is different—storage vs. communication), as well as the granularity, which is a lot finer in our case and takes frame and macroblock dependences into account. To the best of our knowledge, this is the first work that applies unequal error correction to video storage. Our methodology could be also applied to video streaming as well, where different bits can be transferred through network channels of different reliability, as well as approximate video processing, where less important video bits may need less precise computing.

Efforts to support approximate storage or memory in DRAM have focused on reducing the frequency of DRAM refresh operations to save energy at the expense of errors [8, 9]. Ranjan et al. leveraged several mechanisms in STT-MRAM cells that achieve disproportionate energy improvements (40%) at the cost of rare errors and proposed quality-aware ISA extensions for load/store operations. These proposals could benefit from adequate programming language support for approximate data types [16]. However, they would still require the programmer to manually specify the reliability requirements of their data. Our work seeks to automatically determine the bit-level reliability requirements for a particuar data type and would be a great fit for such systems.

10. Conclusion

Videos remain the most voluminous data type today despite the effectiveness of video compression. While storing multiple bits per cell can significantly increase the storage density, it exposes high error rates that quickly grow with the achieved gains, producing unacceptable noise levels for encoded videos. In this work we proposed a novel and efficient methodology that computes bit-level reliability requirements for encoded videos by tracking visual and metadata dependencies within the encoded bit stream, and show how reliability of video storage can be traded for density in an optimal way. By optimally balancing the total noise, which includes both the compression-related deterministic noise and the approximation-related non-deterministic noise, against the overall density benefits provided by both, we achieved quality/density points that neither compression nor approximation could achieve alone. When applied to a dense but error-prone PCM substrate with eight information levels per cell, the proposed methodology reduces the error correction overhead by half in the worst case—i.e., under the most error-intolerant encoder settings. Furthermore, we have shown that the state-of-the art encryption schemes can be applied to the approximate storage in a way that compromises neither privacy nor quality of encoded videos.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful feedback. This work was partially supported by the Swiss National Science Foundation under grant P2ELP2_161823, by the National Science Foundation under grant CCF-1518703, by the DARPA BRASS program under grant FA8750-16-2-0032, by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and by gifts from Intel.

References

[1] High efficiency video coding: Recommendation ITU-T H.265. https://www.itu.int/rec/T-REC-H.265.

[2] Advanced video coding for generic audiovisual services: Recommendation ITU-T H.264. https://www.itu.int/rec/T-REC-H.264.

[3] VideoLAN x264 library and application. http://www.videolan.org/developers/x264.html.

[4] F. Frescura, M. Giorni, C. Feci, and S. Cacopardi. JPEG2000 and MJPEG2000 Transmission in 802.11 Wireless Local Area Networks. *IEEE Transactions on Consumer Electronics*, 49 (4):861–871, Nov. 2003.

[5] Q. Guo, K. Strauss, L. Ceze, and H. Malvar. High-Density Image Storage Using Approximate Memory Cells. In *In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[6] Z. Guo, Y. Nishikawa, R. Y. Omaki, T. Onoye, and Shirakawa. A Low-Complexity FEC Assignment Scheme for Motion JPEG2000 over Wireless Network. *IEEE Transactions on Consumer Electronics*, 52(1):81–86, Feb. 2006.

[7] J. Kastrenakes. Google Announces Unlimited Pictures and Video Storage with new Photos App. http://www.theverge.com/2015/5/28/8678629/google-photos-app-announced, 2015.

[8] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning. In *In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[9] J. Lucas, M. Alvarez-Mesa, M. Andersch, and B. Juurlink. Sparkk: Quality-Scalable Approximate Storage in DRAM. In *The Memory Forum*, 2014.

[10] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[11] D. Marpe, H. Schwarz, and T. Wiegand. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):604–632, July 2003.

[12] L. Merritt and R. Vanam. Improved Rate Control and Motion Estimation for H.264 Encoder. In *IEEE International Conference on Image Processing*, 2007.

[13] Multimedia Signal Processing Group. VQMT: Video Quality Measurement Tool. http://mmspg.epfl.ch/vqmt.

[14] National Institute of Standards and Technology. Advanced Encryption Standard (AES). http://www.nist.gov/, 2001.

[15] I. Richardson. *The H.264 Advanced Video Compression Standard*. John Wiley and Sons, 2nd edition, 2010.

[16] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate Data Types for Safe and General Low-power Computation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011.

[17] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate Storage in Solid-State Memories. In *In Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2013.

[18] Solid State Storage Initiative. NAND Flash Solid State Storage for the Enterprise: An In-depth Look at Reliability. `http://www.snia.org/sites/default/files/SSSI Reliability White Paper 4-09 B.pdf`.

[19] Team FanBridge. YouTube Trends of 2016 (So Far). https://www.fanbridge.com/blog/youtube-trends-of-2016, 2016.

[20] The Netflix Tech Blog. Toward A Practical Perceptual Video Quality Metric. http://techblog.netflix.com/2016/06/toward-practical-perceptual-video.html, 2016.

[21] S. Winkler and P. Mohandas. The Evolution of Video Quality Measurement: From PSNR to Hybrid Metrics. *IEEE Transactions on Broadcasting*, 54(3):660–668, Sept. 2008.

[22] Xiph.Org Foundation. Xiph.org Video Test Media. https://media.xiph.org/video/derf/.