Long-Term JPEG Data Protection and Recovery for NAND Flash-Based Solid-State Storage

Yu-Chun Kuo, Ruei-Fong Chiu, and Ren-Shuo Liu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan
{*yuchunkuo, rfc*}*@gapp.nthu.edu.tw, renshuo@ee.nthu.edu.tw*

***Abstract*—NAND flash memory is widely used in solid-state storage including SD cards and eMMC chips, in which JPEG pictures are one of the most valuable data. In this work, we study NAND flash memory-aware, long-term JPEG data protection and recovery. Our goal is to increase the robustness of JPEG files stored in flash-based storage and rescue JPEG files that are corrupted due to long-term retention. JPEG files with our proposed protection techniques are compatible with existing JPEG viewers. We conduct real-system experiments by storing JPEG files on 16 nm, 3-bit-per-cell flash chips and let the JPEG files undergo a retention process equivalent to ten years at 25°C. Experimental results show that the proposed techniques can rescue corrupted JPEG files to achieve a PSNR improvement of up to 23.5 dB.**

***Keywords*-flash memories; nonvolatile memory; image storage; image restoration; fault tolerance; error correction;**

I. INTRODUCTION

NAND flash memory (flash for short) is widely used as mass storage devices such as removable mass storage devices (e.g., SD cards) and embedded mass storage devices (e.g., eMMC and UFS chips in smartphones). More than one billion SD cards and one billion eMMC chips are sold per year [2], [6], which constitute massive storage of tens to hundreds of exabytes. Thus, unsurprisingly, each of us owns multiple these mass storage devices. Within these flash-based storage devices, JPEG pictures are one of the most valuable data.

Interestingly, many people usually do not realize or forget that flash-based storage is intrinsically not good at *long-term data preservation*, and thus leaving valuable pictures in SD cards and smartphones for a long time (e.g., a few years) can risk uncorrectable errors, which result in corrupted pictures. The reason why uncorrectable errors occur is that flash memory is naturally prone to have retention errors. As the fabrication process shrinks and multiple bits are stored in a cell, nowadays flash memory can exhibit a bit error rate (BER) as high as 1% after data are stored in flash for a few years, which can exceed the error correction strength of flash-based storage. This issue is even worse for SD cards and eMMC chips in smartphones because they may be left without power to perform periodically scrubbing for multiple years.

In this work, we study *long-term, flash-aware JPEG data protection and recovery*. The goal is to increase the robustness of JPEG files stored in flash-based storage and rescue JPEG files that are corrupted due to long-term retention errors. We make several observations as follows. First, the header part of JPEG files is more critical than the remaining part. Second, the reliability of flash-based storage is extremely skewed across its entire capacity: strong pages can retain data much longer than weak pages do. Third, a single bit error can severely deteriorate the quality of a JPEG image owing to two kinds of *error propagation phenomena*, which we will explain shortly in Section III-A. Lastly, we observe a chance to brute-forcibly find the locations of bit errors according to the JPEG decoding procedure.

Based on the above observations, we propose the following four techniques: 1) Strong-Page Header Protection, 2) Bit Error Propagation Prevention, 3) DC Error Propagation Mitigation, and 4) Huffman-Assisted Error Correction. Note that even with our proposed techniques, JPEG files (without uncorrectable errors) can still be decoded by existing JPEG viewers as usual. This compatibility feature is crucial because JPEG viewers have already been widely adopted.

The rest of this paper is organized as follows. Section II provides the background of JPEG image compression. Section III presents our observed error propagation phenomena and our proposed four techniques. Section IV evaluates our proposed techniques, Section V presents related works, and Section VI concludes this work.

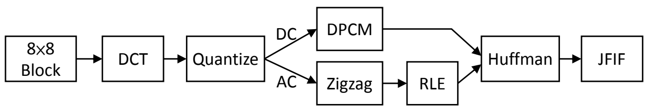
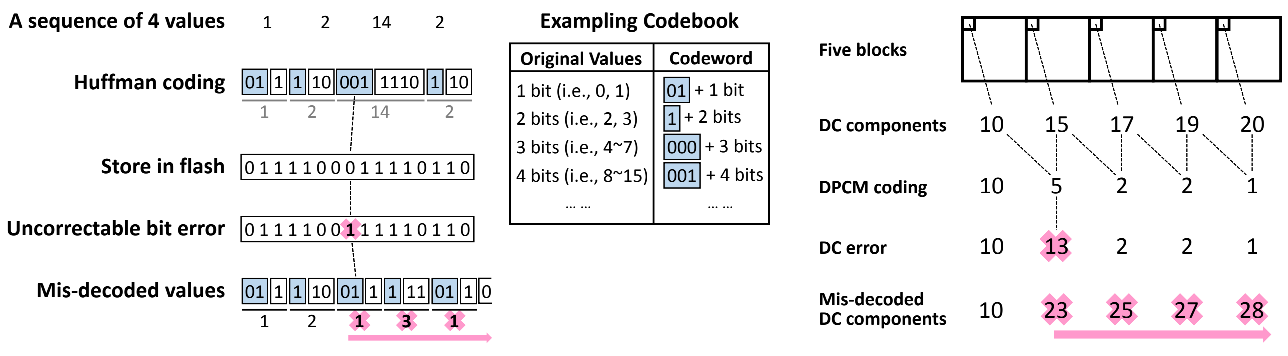


Figure 1: Encoding procedure of JPEG

II. BACKGOURND

JPEG (Joint Photographic Experts Group) is the most widely used image format by digital cameras and smartphones. Although JPEG was born in 1992, and several alternatives to JPEG have been proposed since then, including JPEG-2000 and JPEG-XR, they all fail to replace JPEG [12].



(a) Bit error propagation (b) DC error propagation

Figure 2: Two error propagation phenomena

As shown in Figure 1, to encode an image as a JPEG file, encoders first divides the image into multiple blocks of 8×8 pixels. Each block is processed by the discrete cosine transform (DCT), which transforms the block from the spatial domain to the frequency domain. The output of DCT is also a block of 8×8 values. The upper-left most value corresponds to the lowest frequency component (also known as the DC component) of the 8×8 block, and the bottom-right most value corresponds to the highest frequency component.

After the DCT, JPEG encoders leverage four schemes to achieve image compression: quantization, differential pulse code modulation coding (DPCM), run-length encoding (RLE), and Huffman entropy coding. As for quantization, since human visual systems emphasize the low-frequency (DC) components of images than the high-frequency (AC) components, JPEG encoders quantize the values of AC components using coarse steps to reduce their information. Regarding DPCM coding, the DC components of a row of consecutive 8×8 blocks are treated as a sequence, and JPEG encoders record the differences between each two consecutive DC components. Since the color and brightness of meaningful images typically exhibit smoothness, keeping the differences (i.e., relative values instead of absolute values) achieve compression. Regarding RLE, the 63 AC components of each 8×8 block is linearized according to a zigzag order, and every consecutive run of zeros are represented using only a run-length number. Regarding Huffman entropy coding, each DC component and each non-zero AC component with the length of its leading zeros are treated as a symbol, respectively. Symbols that have a high occurrence frequency are encoded using shorter codewords, and symbols that have a lower occurrence frequency are encoded using longer codewords.

The compressed data after Huffman coding are referred to as bitstreams. A JPEG file typically encapsulates bitstreams according to the JPEG File Interchange Format standard (JFIF). The JFIF standard defines various kinds of markers to separate different contents in a JPEG file such as the SOI (Start of Image), DQT (Define Quantization Table), DHT (Define Huffman Table), APP (Application data), and COM (Comment) markers.

III. Observations and Design

We first describe the observed *error propagation phenomena*, which severely deteriorate the image quality of JPEG files that have uncorrectable errors. Then we explain the *skewed storage reliability phenomenon*, which is observable in flash-based storage but not in hard disk drives (HDDs). We propose the following four techniques: 1) Strong-Page Header Protection, 2) Bit Error Propagation Prevention, 3) DC Error Propagation Mitigation, and 4) Huffman-Assisted Error Correction.

A. Error Propagation Phenomena

Figures 2(a) and 2(b) illustrate the two observed error propagation phenomena. We refer to the first phenomenon as *Bit Error Propagation*, which results from the fact that Huffman codes belong to variable-length codes. Decoding variable-length codes heavily relies on correctly decoding the length of the codes, and a bit error can mess up things. Let us take Figure 2(a) for example. Four values (1, 2, 14, and 2) are encoded into four pieces of bits (011, 110, 0011110, and 110) according to the exampling codebook in the figure. During decoding, the starting 01 unambiguously implies the existence of a single following bit, 1, according to the codebook. Similarly, the fourth bit, 1, unambiguously implies two following bits, 10. However, a bit error (the eighth bit, $0 \rightarrow 1$) causes the length of the third code to be incorrectly decoded (7-bit 0011110 $\rightarrow$ 3-bit 011). It not only interferes with the third value ($14 \rightarrow 1$); the incorrect length further causes the fourth and likely many following values to be incorrectly decoded, too ($2 \rightarrow 3$, etc.).

The second error propagation phenomenon is *DC Error Propagation*, which results from the fact that JPEG utilizes DPCM codes to compress the DC components of a sequence

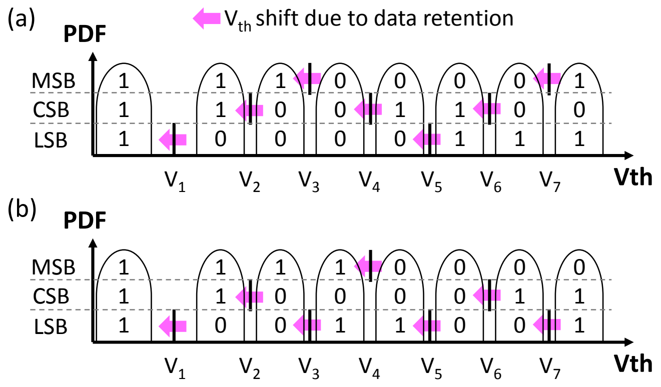

(a)
V_{th} shift due to data retention
PDF
MSB
CSB
LSB
1 1 1 0 0 0 0 1
1 1 0 0 1 1 0 0
1 0 0 0 0 1 1 1
V_1 V_2 V_3 V_4 V_5 V_6 V_7 Vth
(b)
PDF
MSB
CSB
LSB
1 1 1 1 0 0 0 0
1 1 0 0 0 0 1 1
1 0 0 1 1 0 0 1
V_1 V_2 V_3 V_4 V_5 V_6 V_7 Vth

Figure 3: Two typical data-V_{th} mappings of TLC flash

of consecutive 8 × 8 blocks. Figure 2(b) illustrates five consecutive blocks whose DC components are 10, 15, 17, 19, and 20, respectively. According to DPCM codes, the five values are encoded as the first absolute value (10) and four relative values (5, 2, 2, and 1). For example, if a bit error interferes with the second DC value (5→13), a DPCM decoder not only produces an erroneous second DC value (15→23). Moreover, the following third, fourth, and fifth DC values also become incorrect (17→25, 19→27, and 20→28).

B. Skewed Storage Reliability

Unlike traditional HDD storage, flash-based storage exhibits a unique property that its storage reliability varies dramatically depending on page addresses: some pages can store data much more reliably than the others. We referred to this property as *Skewed Storage Reliability*. The potential sources of skewed storage reliability include process variations and storing multiple bits per cell. The latter is consistent among flash chips of the same part number, and this work exploits it.

Figure 3 shows two typical data-V_{th} mappings of flash memory with three bits per cell (also referred to as TLC). Flash stores different data using different threshold voltages (V_{th}'s), and TLC utilizes eight different V_{th}'s. If the V_{th} of a cell is lower than V_1, its stored three bits are 111, which is also the erased state of flash; if V_{th} is between V_1 and V_2, the three bits are 110, and so on. Flash is prone to retention errors, which are caused by V_{th} shift over retention time. For example, if the V_{th} of a flash cell shifts and crosses V_1, data 110 are incorrectly interpreted as 111, and in this case, the least significant bit of the cell becomes erroneous.

Figure 3 also can explain the reasons why storing three bits in a cell results in skewed storage reliability. In common practices, flash vendors separate the three bits of each cell into three flash pages. They are widely referred to as least-significant bit (LSB) pages, center-significant bit (CSB) pages, and most-significant bit (MSB) pages. Figure 3(a) illustrates the mapping of the flash we experiment. One can see that among seven V_{th} boundaries (V_1 to V_7), crossing one of three boundaries, V_2, V_4, and V_6, leads to a CSB error. In comparison, there are only two boundaries related to LSB and MSB errors, respectively. Therefore, CSB pages are more sensitive to V_{th} shift than LSB and MSB pages do. In addition, the lowest V_{th} state (111) typically separates far from the second V_{th} state (110) because the former is the erased state. Therefore, the chance that V_{th} shifts and crosses V_1 is relatively low. Owing to the above reasons, LSB pages are the strongest type of pages, and CSB pages are the weakest. For flash designed according to Figure 3(b), previous characterization works have shown that MSB pages are the strongest type of pages [4], [15].

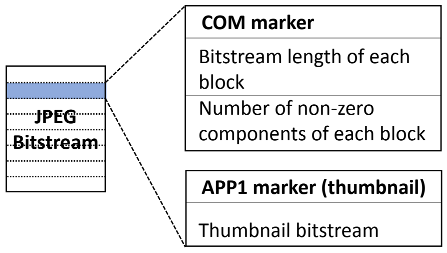

JPEG Bitstream
COM marker
Bitstream length of each block
Number of non-zero components of each block
APP1 marker (thumbnail)
Thumbnail bitstream

Figure 4: Extra header information

C. Strong-Page Header Protection

The importance of JPEG contents is not homogeneous across a JPEG file. For example, the JFIF standard formats a JPEG file into header and data sections. The header section comprises critical information for correctly decoding the JPEG file, including image dimensions, sampling factors, quantization tables, and Huffman tables. In comparison, the data section comprises the bitstream of each 8×8 block. Therefore, the header section is more critical than the data section.

According to the above observations, we propose to allow JPEG applications to allocate critical data to strong pages. This feature can be implemented by adding vendor commands to the command interface of SD cards and eMMC chips, and the modification should be moderate and practical. We leave prototyping such flash-based storage as our future work.

D. Bit Error Propagation Prevention

As mentioned earlier (Figure 2(a)), a bit error in the data section of a JPEG file not only interferes with the block that contains the bit error. Even worse, the bit error usually continuously fails the decoding of the successive blocks. To address this issue, as illustrated in Figure 4, we propose to additionally store the bitstream length of each block in the header section of JPEG files (in strong flash pages). Specifically, we store such information in COM markers in the header section. By doing so, the augmented JPEG file is still fully compatible with existing image viewers because they ignore unrecognized contents in COM markers.

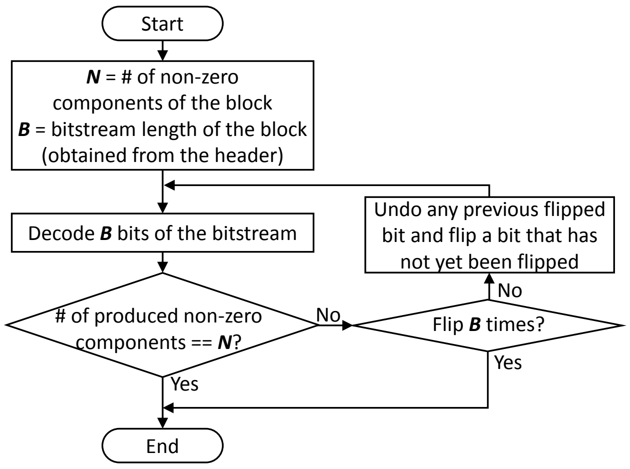

Start
N = # of non-zero components of the block
B = bitstream length of the block (obtained from the header)
Decode B bits of the bitstream
Undo any previous flipped bit and flip a bit that has not yet been flipped
# of produced non-zero components == N?
No
Flip B times?
No
Yes
Yes
End

Figure 5: Huffman-assisted error correction

Once uncorrectable errors occur in the JPEG file, an image viewer that adopts our proposed JPEG recovery techniques can recognize the information in the COM marker and utilize it to rescue the JPEG file.

E. DC Error Propagation Mitigation

The DC components of JPEG images are essential to image quality. As discussed earlier, even with *Bit Error Propagation Prevention*, DC error propagation can still occur and deteriorate the quality of JPEG images. To address this issue, we propose to mitigate the degree of DC error propagation by judiciously storing duplicated DC information in the JPEG header section.

More specifically, as illustrated in Figure 4, we propose to leverage the existing thumbnail in the APP1 marker in the header section. JPEG encoders typically embed a thumbnail JPEG in the APP1 marker of the main JPEG file to facilitate image preview. We propose to set the width and height of the thumbnail to be one-eighth that of the main JPEG, respectively. By doing so, each pixel of the thumbnail can be treated as the DC component of the corresponding 8×8 block. Once a DC component decoded from the main JPEG deviates too much from the corresponding pixel value of the thumbnail, our decoder uses the latter (i.e., thumbnail pixel) as the DC component instead of using the former, and thus the degree of DC error propagation is mitigated.

F. Huffman-Assisted Error Correction

The Huffman bitstream of each 8×8 block is roughly 100 bits on average. Even if the BER of flash is as high as 1%, a significant portion of 8×8 blocks contain only single erroneous bit. Therefore, there is a chance to find the location of the bit error by trial-and-error. More specifically, decoders can brute-forcibly try to flip one bit at a time and repeatedly decode a block until valid results are obtained. Note that Huffman-Assisted Error Correction cannot work independently without Bit Error Propagation Prevention mentioned earlier because without the bitstream length of each block, once a block contains more errors and is uncorrectable, decoders can fail to locate the correct starting boundary of other successive blocks.

We propose to additionally record the number of non-zero components of each 8×8 block in COM markers in the header section (Figure 4). As shown in Figure 5, when decoding an 8×8 block, a JPEG decoder can determine whether the obtained results are invalid: if the number of non-zero components produced by the decoding process does not match the recorded number, the results are invalid, and the decoder can start the brute-force bit flipping procedure. Note that this procedure does incur extra latency, but since this procedure is for rescuing corrupted JPEG files, decoding speed is not a top-priority concern. In addition, there are some chances of false positive, i.e., the number of non-zero components matches, but the block still contains errors, but some false positives are also acceptable for rescuing corrupted JPEG files.

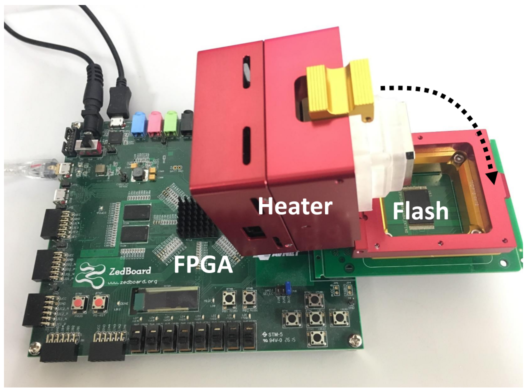


Figure 6: Setup of an FPGA, a heater, and flash under test

IV. Evaluation

A. Experimental Setup

We conduct experiments using a Xilinx Zedboard FPGA platform, 16 nm TLC flash chips, and a heater as shown in Figure 6. We design digital circuits and firmware for the FPGA to enable us to read, write, and erase the flash chips using the FPGA. We program and erase flash for 400 times before storing JPEG files in it.

A total of 105 JPEG files are used in the experiments. One hundred of them are of the size of $3,264 \times 2,488$, generated by an iPhone smartphone of one of the authors. Another five JPEG files are of the size of $3,072 \times 2,048$ from the Kodak PhotoCD [1]. We add protection information into the JPEG files and write them on flash chips with JPEG headers allocated to strong flash pages (i.e., LSB pages).

To study the data retention impacts, we let the JPEG files to undergo multiple years of retention time. The JEDEC

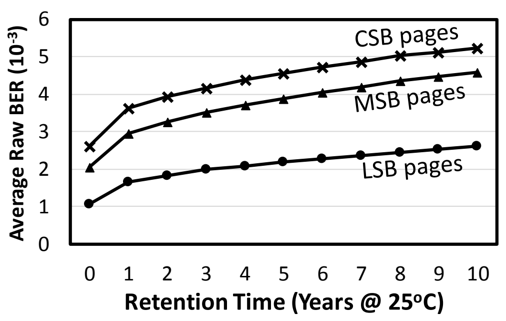


Figure 7: Average raw BERs of the TLC pages

JESD 47H.01 industrial standard provides a temperature-accelerated method to realistically perform such an experiment: by heating flash chips to 85°C, the effective data retention time is $1,300\times$ longer that under 25°C. According to this method, we heat flash chips by 85°C for seven hours each time (equivalent to one year under 25°C) and retrieve the JPEG files from the flash, which exhibit raw bit errors. We repeat this process until 70 hours, which are equivalent to ten years under 25°C.

The error correction codes (ECC) of flash-based storage differ from vendors to vendors and are not publicly available. As a reasonable assumption, we view each 8 KB flash page as four interleaved 2 KB codewords and assume that ECC can correct codewords with a BER up to 5×10^{-3}. For codewords that exhibit a BER higher than 5×10^{-3}, we assume that the bit errors are uncorrectable and remain in JPEG files.

We set the baseline configuration as follows. JPEG files are stored in flash storage as usual without using our techniques. One exception is that we conservatively assume that the header sections of JPEG files are allocated to strong pages for the baseline. This experimental setup favors the baseline because the baseline should be oblivious to flash memory.

We use OpenCV to read JPEG files as the baseline setting. In comparison, we extend a JPEG decoder [3] with our proposed techniques for rescuing corrupted JPEG files. PSNR (peak signal-to-noise ratio) and SSIM (structural similarity index) [14] metrics are used to quantify the effectiveness of our proposed techniques. In common practices, images with PSNR greater than 30 dB or SSIM greater than 0.9 are considered to have high quality.

B. Experimental Results

Figure 7 shows the average raw BER of JPEG files at different retention time. We separately calculate the BERs of three types of pages. Over a (temperature-accelerated) ten years course, CSB pages consistently exhibit the highest average BER, and LSB pages consistently exhibit the lowest average BER.

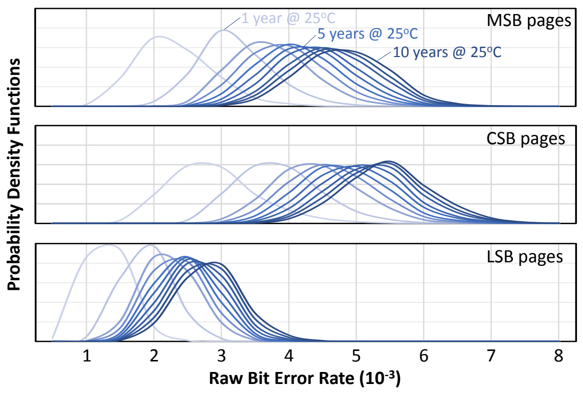


Figure 8: Distributions of the raw BERs of the TLC pages

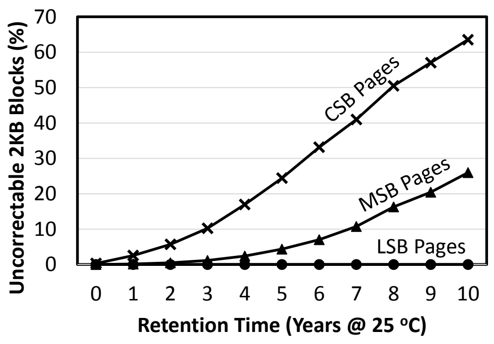


Figure 9: Average percentage of uncorrectable 2 KB codewords

Figure 8 further shows the detailed distributions of flash BERs at different retention time. In general, BERs follow bell-shape distributions and increase over time. CSB pages are the weakest, and the center of the BER distribution can reach up to 5.5×10^{-3}. In comparison, LSB pages are the strongest, and the center of the BER distribution only reaches 2.9×10^{-3}

Figure 9 plots the percentage of uncorrectable 2 KB codewords over time. Again, CSB pages are the weakest, and they start to exhibit uncorrectable errors since a very early stage (e.g., 0.8% at the beginning) and the percentage raises quickly to 64% in ten years. Note that although the percentage of uncorrectable errors can be reduced by increasing the strength of ECCs, the relative reliability among LSB, MSB, and CSB pages of this flash does not change: CSB pages (i.e., weak pages) always exhibit errors earlier than LSB pages (i.e., strong pages).

Figure 10 visualizes the recovery results of a JPEG file that is corrupted due to 10-year retention. In the baseline configuration, even though we allocate JPEG header sections to strong pages, the image is still heavily corrupted. This outcome mainly results from bit error propagation. With the Bit Error Propagation Prevention technique, the SSIM

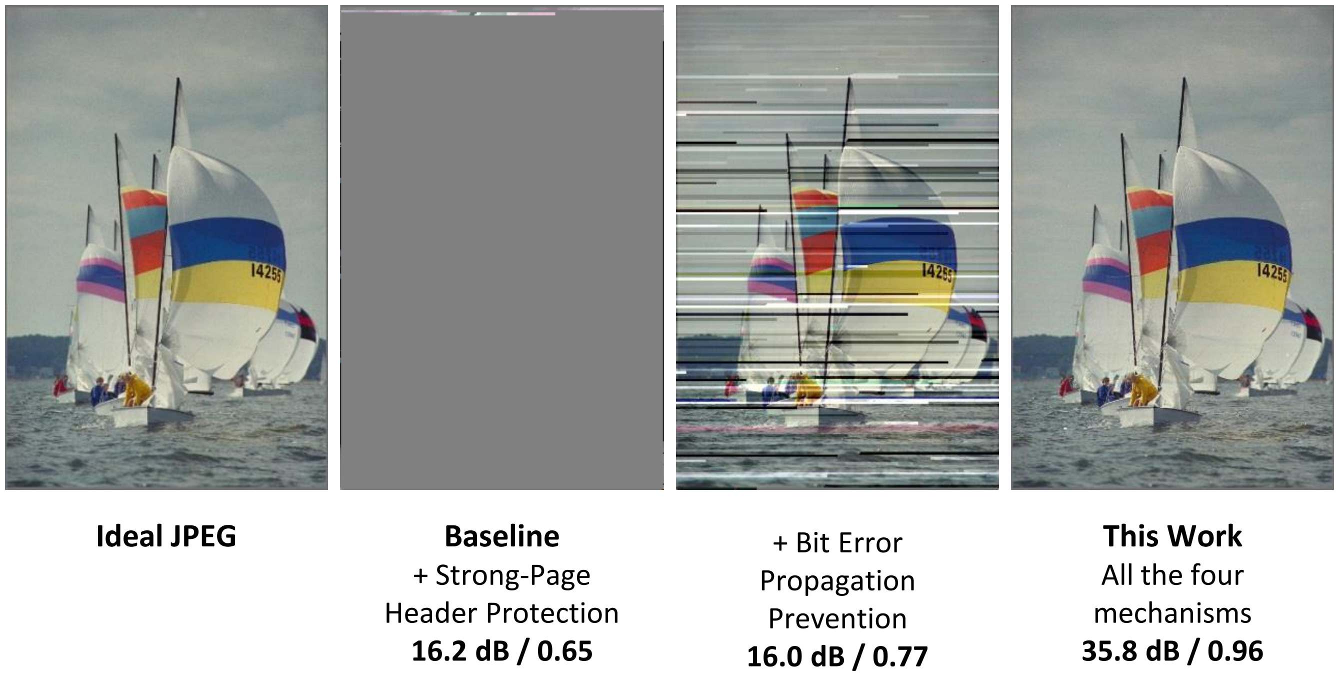


Figure 10: Recovery results of a corrupted (10-year) JPEG file and PSNR/SSIM results

raises to 0.77, and the picture quality is better but still heavily impacted by observable horizontal stripes, which result from DC error propagation. By adopting all the proposed techniques, the quality of the resulting image is largely improved (PSNR=35.8 dB, and SSIM=0.96).

Figure 11 shows the averaged PSNR over the 105 JPEG files. In the baseline configuration, as long as flash-based storage exhibits a slight number of uncorrectable bit errors, the PSNR severely degrades. In comparison, the proposed techniques can recover corrupted JPEG files effectively. For example, when flash exhibits uncorrectable bit errors, the PSNR of the baseline drops to lower than 30 dB and saturates at 11 dB. As shown in Figure 10, an image with a 16 dB PSNR is already unacceptable, not to mention images with an even worse PSNR. In comparison, the proposed techniques keep the average PSNR greater than 31 dB within 70 hours of 85°C baking (equivalent to 10 years retention at 25°C). The maximum PSNR improvement is 23.5 dB at the third year.

Figure 12 plots the SSIM results. The trend is similar to that of PSNR. The proposed techniques consistently outperform the baseline. Note that although the baseline seems to be able to keep the SSIM at around 0.57 over 10-year retention, such SSIM is already unacceptable as shown in Figure 10.

C. Storage and Latency Overhead

The storage overhead for supporting the proposed techniques is 9.9% in total. The original size of the tested 105 JPEG files is 201 MB, and that with the proposed extra protection information (the COM and APP1 markers) increases to 220 MB. With the protection capability shown in the above experiments, we anticipate that many users are willing to pay 9.9% storage overhead.

The 9.9% storage overhead can be otherwise used for extra ECC parities, which can correct more raw bit errors than our techniques because ECCs are mathematically optimized. However, it is noteworthy that there are some concerns about employing extra ECC parities. First, the per-bit cost of SD cards and eMMC chips are sensitive, so flash and storage vendors are likely reluctant to pre-allocate space for storing extra ECC parities. Second, if SD cards and eMMC chips dynamically allocate extra ECC parities at run time, their visible capacity also changes at run time, which may be problematic to some users, applications, and operating systems. Third, employing extra ECC parities at the application level is less effective because modern ECCs of flash storage such as low-density parity check (LDPC) rely on low-level access to flash cells. In comparison, the proposed techniques are pure application-level and do not cause the above problems.

Our program takes 11.9 seconds on average for recovering a corrupted (10-year) JPEG file. Note that decoding speed is not a top-priority concern for rescuing corrupted JPEG

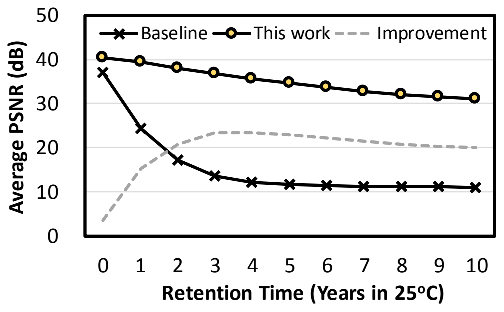


Figure 11: PSNR improvement over 10 years at 25°C

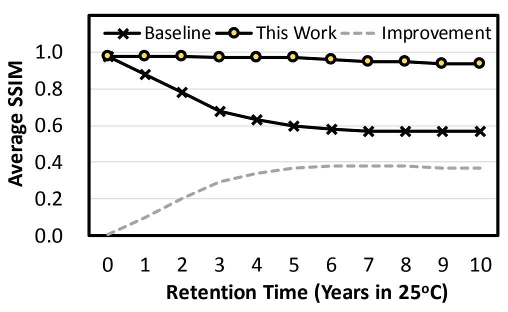


Figure 12: SSIM improvement over 10 years at 25°C

files. In addition, it is easy to parallelize the recovery tasks of multiple corrupted JPEG files using multi-core PCs or servers. Lastly, we have not optimized the speed of our program yet, so there is still a speedup headroom.

V. Related Works

This work is the first on long-term JPEG data protection and recovery with backward compatibility. Previous works on approximate and retention-relaxed storage are related but fundamentally different. For example, [5], [10], [13] and [16] consciously lower image quality to increase storage performance and density, but none of them focuses on the most widely used JPEG format, and none of them proposes strategies to rescue compressed images with high uncorrectable BERs. Specifically, [13] and [16] focus on RAW images, [5] focuses on JPEG-XR, and [10] focuses on JPEG-2000. In addition, [7]–[9], [11] propose to relax data retention to increase the performance and lifetime of nonvolatile memory, but their design is for server workloads instead of JPEG files.

VI. Conclusions

This work focuses on increasing the robustness of JPEG files stored in flash-based storage (i.e., protection) and rescue corrupted JPEG files using the protection (i.e., recovery). Four innovative techniques are proposed: 1) Strong-Page Header Protection, 2) Bit Error Propagation Prevention, 3) DC Error Propagation Mitigation, and 4) Huffman-Assisted Error Correction. The proposed techniques incur 9.9% storage overhead on average and can help to rescue corrupted JPEG files to achieve 23.5 dB better PSNR on average.

Acknowledgement

We thank the anonymous reviewers for their valuable feedback. This work is supported in part by the Ministry of Science and Technology (MOST) of Taiwan under grants 107-2218-E-007-034, 107-2218-E-007-031, and 107-2218-E-007-001. We also thank the National Center for High-performance Computing (NCHC) of Taiwan for computer time and facilities.

References

[1] Kodak PhotoCD (www.math.purdue.edu/~lucier/PHOTO_CD/).

[2] The state of the microSD memory card in 2016 (www.sdcard.org), 2016.

[3] bkenwright. loadjpg.cpp (www.xbdev.net).

[4] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proc. IEEE*, 105(9):1666–1704, Sep. 2017.

[5] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar. High-density image storage using approximate memory cells. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[6] IHS Markit. A tale of two memories: NAND eMMC hits a record year, while NOR flash shrinks further, 2014.

[7] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang. NVM Duet: Unified working memory and persistent store architecture. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[8] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen. DuraCache: A durable SSD cache using MLC NAND flash. In *Design Automation Conference (DAC)*, 2013.

[9] R.-S. Liu, C.-L. Yang, and W. Wu. Optimizing NAND flash-based SSDs via retention relaxation. In *File and Storage Technologies (FAST)*, 2012.

[10] Z. Liu, T. Liu, J. Guo, N. Wu, and W. Wen. An ECC-free MLC STT-RAM based approximate memory design for multimedia applications. In *IEEE Symposium on VLSI (ISVLSI)*, 2018.

[11] Y. Pan, G. Dong, Q. Wu, and T. Zhang. Quasi-nonvolatile SSD: Trading flash memory nonvolatility to improve storage system performance for enterprise applications. In *High-Performance Computer Architecture (HPCA)*, 2012.

[12] P. Rhodes. Why is it so hard to replace JPEG? (www.redsharknews.com/technology/item/6145-why-is-it-so-hard-to-replace-jpeg), 2019.

[13] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. In *International Symposium on Microarchitecture (MICRO)*, 2013.

[14] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, April 2004.

[15] E. Yaakobi, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf. Characterization and error-correcting codes for TLC flash memories. In *International Conference on Computing, Networking and Communications (ICNC)*, 2012.

[16] H. Zhao, L. Xue, P. Chi, and J. Zhao. Approximate image storage with multi-level cell STT-MRAM main memory. In *International Conference on Computer-Aided Design (ICCAD)*, 2017.