On the Theory and Practice of Data Recovery with Multiple Versions

Michael Mitzenmacher
Division of Engineering and Applied Sciences
Harvard University
Email: michaelm@eecs.harvard.edu

***Abstract*— Replication and error-correction are standard mechanisms to cope with data corruption. We consider how these two approaches can be combined for preservation of digital data. We provide an information-theoretic model and theoretical framework, and discuss how recent breakthroughs such as the development of Raptor codes may play a part in future practical systems.**

I. Introduction

The problem of coping with document deterioration is well-known in the library sciences. While one would hope that shift to digital media would alleviate these problems, digital copies can similarly deteriorate over time [7], [11]. While estimated lifetimes of many digital media are a complicated subject with significant debate, estimates of the life expectancy of CDs are generally reported to be between five and fifty years, and for magnetic tapes the numbers are between ten and twenty years. Data recovery of digital media will likely continue to be an ongoing problem.

Error-correction and replication are both common techniques for enabling data recovery. Replication is often used to deal with loss of complete copies, for example due to natural disasters or system failures. Error-correction is more commonly used with a single copy of data, allowing for data protection. In this paper, we attempt to develop a systematic framework for considering what can be accomplished using both techniques in conjunction with each other. This idea has appeared in alternative guises before; certainly multiple-source coding problems, including many variations of multiple description source coding (see, e.g. [2]), have been widely considered, and this problem shares some characteristics with this previous work. Our setting, however, appears different, as we now clarify.

We consider the problem of using error-correction effectively in situations where there are multiple *versions* of an archived data file. By a version, we mean a possibly encoded representation of the file; versions need not be identical (and hence we avoid using the more ambiguous term copies). At some later time, the data is desired, but by this time each individual version may have deteriorated, leading to bit errors. Further, some versions may have entirely disappeared. The versions that remain will be used, with the goal to reconstruct an error-free copy of the original data. The type and quantity of errors is not known in advance. Here, we focus on the case where bit errors are independent in each version, and the error rate for each version is not known in advance, as this allows a sound theoretical formulation, but other error models could also be considered similarly. Also, to be clear, we ignore the often pressing practical question of whether possibly obsolete hardware and software resources remain available to read the data; our focus is entirely on the information-theoretic issue of what to do when faced with bit-level errors.

What types of error-correction should be used in this situation? We consider natural variations of this problem, establish the basic theory by determining capacity bounds in somewhat idealized models, and discuss the issues related to code design. While we do not develop specific coding solutions here, we focus on the feasibility of the following optimistic goal: efficient encoding and decoding schemes that are simultaneously near-optimal in terms of error rates handled, regardless of the number of versions recovered. We believe that new developments in the area of digital fountain codes, specifically work on Raptor codes, suggests this goal may be feasible. Indeed, we believe two contributions of this paper are to suggest a novel application for Raptor codes and to suggest several specific design problems for Raptor codes that would be useful in this context.

Our problem appears different from previous work in that we simultaneously desire complete recovery (no errors), no advance knowledge of error rates or even the number of available versions, near optimal performance, and practical viability.

As a starting point, we note that the obvious solution is to not use any error-correction at all, but use identical versions of the data, and reconstruct each bit by taking the majority vote among all versions. This approach can be effective when the bit error-rate is very small, the number of versions is large, and the total number of bits is moderate. The approach is obviously rather limited. For example, if each bit is independently corrupted with probability 0.001 and there are five versions, the bit error rate using the majority rule is roughly 10^{-8}. While this error rate would be a substantial improvement, it would still lead to several bit errors for a standard CD worth of data.

The majority-vote idea may seem overly simple, but a similar voting-based mechanism (at a block, not bit, level) is used in the notable LOCKSS (Lots of Copies Keep Stuff Safe) peer-to-peer preservation system [3]. The voting system requires relatively frequent communication among document collections. On the other hand, the LOCKSS system has much stronger requirements than what we consider here, as it is meant to cope with possible malicious data corruption as well as data deterioration. We believe this work suggests that basic coding techniques, in conjunction with replication, can yield useful (albeit more basic) digital document protection without a dedicated system. Such ideas may also be useful in systems such as LOCKSS; we leave this for future work.

II. The Version Framework

To model the problem, we assume that we have an initial file F of n bits, and we construct m encoded *versions* of F, each with some number of bits n'. When decoding, we obtain $d \leq m$ encoded versions of F, denoted by $F_1, F_2, \ldots, F_d$. These versions will be subject to bit-level errors. We assume here that in each version errors are independent, with the error rate of the ith version denoted by p_i. Generally we will not know which versions will be available, and indeed we will not even know d in advance. The goal is to design a coding schemes that are nevertheless near optimal for every value of d simultaneously, over as large a range of error rates in the versions as possible.

We say that the versions are *identical* if all encoded versions are initially identical copies. Identical copies are often desirable because they are easier to construct, particularly in cases where the number of original copies produced is significant. We say that the versions are *systematic* if the original data (in unencoded form) is a part of each version. Systematic versions are often desirable if the common case is that the data is to be read without errors. For example, suppose that the data is to be used a great deal in the near future, the risk of errors being introduced in this time frame is small, and the chances of the data being needed again in the distant future, when errors may arise, is relatively small. A systematic version may then be desirable for its faster read times in the common case; encoded versions might require a complete decoding every time the file is to be read. Of course systematic copies are also useful if one expects to want to use or copy parts of the data as well, since generally recovering parts of the data when using encoded data requires computation on the same order as that for full decoding.

There are four natural variations that we focus on here, based on whether the versions are identical and/or systematic. One could also consider other variations; for example, in the case where large number of copies are made, there could be many identical copies of a small number of different versions. As another variation, one could allow each version to consist of a different number of bits n_i. We do not consider such variations here, although they could be studied in our framework.

III. Identical Versions

We start by examining the simpler case of identical versions. Even this case leads to interesting issues and many open problems.

A. Capacity Bounds

We first consider the capacity in the case of identical versions. As an example, suppose that we have two initially identical versions of a data file, and each bit is flipped independently in each copy with probability p. In this case, when trying to reconstruct the data file, a bit will have different values in the two versions with probability $2p(1-p)$. Such bits can be treated as erasures, since we have no indication which value is correct. A bit will take on the wrong value in both versions with probability p^2. Hence, we can think of data as having passed through a channel that erases each bit with probability $2p(1-p)$ and flips each bit with probability p^2; such error/erasure channels have been considered in [4], [8]. Letting X be a data bit and Y be the channel output for that bit, and noting that the capacity is maximized when each input bit is 0 or 1 indpendently with probability $1/2$, we find the capacity $C_{2,p} = I(X,Y)$ is

$$\begin{aligned} C_{2,p} &= 1 - 2p(1-p) - (1-2p(1-p))\log(1-2p(1-p)) \\ &\quad + 2(1-p)^2\log(1-p) + 2p^2\log p. \end{aligned}$$

We note that here and throughout we adopt the convention that that the input is always considered to be n' bits when considering the capacity. As such, our goal is that the capacity is at least n/n' in order for the original data to be recovered.

More generally, consider d initially identical versions of the data file, with each bit being flipped independently with probability p. We can abstract this setting as a channel that takes as input the data file bits $x_1, x_2, \ldots$ and outputs $y_1, y_2, \ldots$, where each y_i is an integer in the range $[0, d]$, corresponding to the number of versions of the file in which the ith bit is a 0. We call this a *binomial channel* with parameter d. (We expect this channel has been studied before, but we have not yet found a reference in the literature.) The capacity $C_{d,p}$ of this channel is

$$C_{d,p} = \\ 1 + \sum_{j=0}^{d} \binom{d}{i} \left(p^i(1-p)^{d-i} \log \frac{p^i(1-p)^{d-i}}{p^i(1-p)^{d-i} + p^{d-i}(1-p)^i} \right).$$

Figure 1 provides a graph of $C_{d,p}$ for $d \leq 5$ and $p \leq 1/2$. It demonstrates a small number of identical versions can greatly improve on the capacity, particularly when the error rate is high. Specifically, we note that when $C_{1,p} = 1 - H(p)$ is small, $C_{d,p} \approx d(1 - H(p))$.

The assumption that all versions have the same error probability is not strictly required to analyze the capacity, although the complexity of the computation can increase significantly without this assumption. If each copy has a different error rate, we can abstract this situation as a channel where the input symbol is a bits and each output symbol is one of the 2^d subsets of $\{1, 2, \ldots, d\}$, corresponding to the set of versions of the file in which the ith bit is a 0. In general it seems it would take time exponential in d to compute the capacity for corruption rates $p_1, p_2, \ldots, p_d$ for the d versions; this may still be reasonably efficient when d is small.

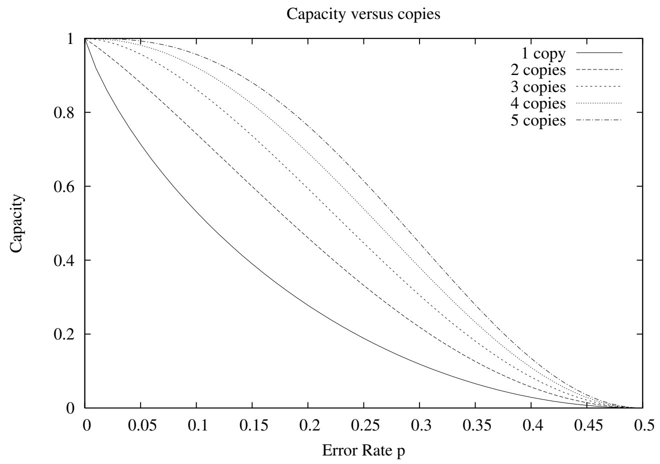

Fig. 1. Capacity curves for varying numbers of initially identical versions with a fixed error rate over all versions.

In practice, we are unlikely to know the error rates for each version in advance, or even at the time of recovery. We consider estimation techniques in the context of coding below.

B. Coding Methods and Design Problems

Our setting appears quite amenable to belief propagation, and in particular to the use of low-density parity-check (LDPC) codes. Let us assume that we can determine the error rate for each version. In this case, the conditional probability of for the value of each bit can be determined, for use in a message-passing belief propagation algorithm.

There are several challenges to problem of code design, however. First, to design the code, one would want to have the density distribution of the initial conditional probabilities for the beliefs associated with a bit. If each version has a distinct error rate, the density distribution of the initial conditional probabilities may not be expressible exactly, as again there are potentially 2^d possible values in the support. If one knows the error rates and d in advance, an approximation to the initial density distribution can be obtained via sampling.

The more vexing challenge is that we simultaneously want codes that perform well for every value of d (or at least for a reasonable range of d). Moreover, the code design should handle as large a range of error rates as possible. If one knew the initial density distribution, one could design a specific code using now standard approaches, but this is not the case here.

A possibility is to apply the technology of Raptor codes. Raptor codes were originally designed to give near-optimal performance, both in terms of capacity and in terms of required computation, over random erasure channels even when the erasure probability is not known in advance. In that setting, a natural model is that the sender will continue to send encoded packets until sufficiently many have been received for decoding to occur. The Raptor code allows encoded packets to be generated individually on the fly without any dependence on the erasure probability, yet achieves near-capacity performance for any value of the erasure probability [9]. Raptor codes have been extended to symmetric binary error channels, where again the goal is to have symbols sent until decoding is possible, without knowing in advance the error probability [6], [10]. More general symmetric channels have also been considered [10]. We note that systematic versions of Raptor codes exist, and that Raptor codes are extremely efficient, and therefore well suited to use in real applications.

Stated another way, with Raptor codes, bits are generated from an unlimited source, and the goal is to decode after receiving as close as possible to the information-theoretically optimal number of bits, regardless of what that number turns out to be. In our setting, we would like instead to successfully decode over as large a space of error rates as possible over all relevant numbers of versions. These problems are, in some sense, duals of each other, making Raptor codes a natural choice for this application.

Under our model, the channel is symmetric, allowing the ideas of [10] to be used. Handling the case where we assume all error rates are equal seems tractable; without further information regarding the possible error rates of individual versions, this also seems the best design choice. We see the design of Raptor-like codes for this setting as an interesting and challenging open problem.

A further challenge for decoding and code design is that the error rates for each version may not be known in advance. The code design must take this into account. Further, for decoding, it is imperative to obtain at least reasonably accurate information regarding the error rates for each version in order to appropriately seed initial probabilities for the belief propagation algorithm. In the standard case of a Raptor code, one can assume the worst case; that is, assume the error rate is the largest it could possibly be while still allowing successful decoding. In the setting with multiple versions, there is no similar worst case; one wants initially to estimate the error rate of each version.

One possibile solution to this estimation problem would be to include test bits with known values, and estimate the fraction of errors based on the results from these test bits. Although this is a reasonable solution, we point out some drawbacks. First, the obvious problem is that test bits represent a waste of space, lowering the rate. For large values of n', however, this cost may not be large, as good estimates of the error rates can be obtained with a fixed number of bits. Second, this approach depends heavily on the underlying model, that each version has a distinct but fixed error rate.

A more interesting approach is to use the received data itself to obtain estimates or bounds on the error rates. (This can also be done in conjunction with using test bits.) Here we make use of the underlying model, but the general idea may be applied to other models as well. To begin, it is instructional to consider the simplest case $d = 2$. The expected fraction of locations where the bits disagree is $p_1(1-p_2)+p_2(1-p_1)$. Hence if $\hat{v}$ is the observed fraction of locations where the bits disagree, we have

$$\hat{v} \approx p_1(1-p_2)+p_2(1-p_1),$$

where standard tail inequalities show the approximation is quite good. The fraction of bit locations where both versions of the bit are flipped is p_1p_2, which for a given value of $\hat{v}$ is maximized when $p_1 = p_2$; assuming $p_1, p_2 < 1/2$, this gives the estimate

$$p_1p_2 \leq \frac{1-\hat{v}-\sqrt{1-2\hat{v}}}{2},$$

which can be pessimistically used for a joint erasure-error decoder. Alternatively, if this error rate exceeds what can be handled by the code, any value up to the maximum possible rate can be tried; the decoding will then generally fail gracefully, yielding a non-codeword. Finally, one could also try multiple runs of the algorithm, using different values of p_1 and p_2 that are consistent with the observed data, to see if any yield a codeword.

For larger values of d, the received data yields enough information to approxiately estimate individual error probabilities. For example, when $d = 3$, if we let $\hat{v}_{12}$ be the fraction of bits where versions 1 and 2 disagree with version 3, we have that

$$\mathbf{E}[\hat{v}_{12}] = p_1p_2(1-p_3) + (1-p_1)(1-p_2)p_3,$$

and similarly there are 3 such equations. Using observed values in place of expectations gives a system of 3 equations and 3 unknowns that can be solved. When $p_1, p_2, p_3 < 1/2$, the system can be solved; if the observed values are sufficiently close to the expectations, and the system of equations is reasonably well behaved (as when p_1, p_2, and p_3 are all sufficiently far from 0), accurate values will be obtained. When $d > 3$, there are in fact more equations than variables and the system might not be solvable exactly when using observed values; instead, maximum likelihood estimates can be computed (or approximate maximum likelihood estimates can be used if desired for speed).

IV. Non-Identical Versions

Using non-identical versions allows significantly more flexibility. In this setting, it seems natural to try to make the value of additional versions as additive as possible. This fits naturally with the abstraction of the digital fountain [1], [5]; one of the main goals of the digital fountain is to allow additional symbols to help decoding in a natural way. This again naturally suggests the use of Raptor codes, as we describe.

A. Capacity Bounds

When versions need not be identical and are not systematic, the problem is as though we have $n'd$ bits, with the bits from the ith version having some error rate p_i. For an appropriate comparison, we again follow a scaling that normalizes by the number of bits per version n', which allows the capacity to be as large as d. The capacity in this case is easily given by

$$d - \sum_{i=1}^{d} H(p_i).$$

Notice that this value can be significantly larger than 1, when $d > 1$; for identical versions, the capacity is bounded above by 1. The potentially large gap between the capacities for identical and non-identical versions demonstrates the large price to be paid for not using something akin to a digital fountain.

When $n' > n$, rates of greater than 1 are of course overkill. If one expects to obtain multiple versions, however, it might be a reasonable design choice to have $n' < n$ with non-identical versions. Indeed, this is akin to using Raptor codes for an error-resilient secret sharing scheme; each version would not be sufficient to reconstruct the original message, but a combination of versions, even assuming errors are introduced, would allow reconstruction if the error rate was not too high.

With the additional constraint that the versions are systematic, the resulting channel is a mixture of the binomial channel defined above for the systematic bits and a simple error channel for each of the versions on the remaining bits. The capacity therefore depends on the ratio n/n'. In the case where all p_i equal p, it is given by

$$\left(1 - \frac{n}{n'}\right) d(1 - H(p)) + \frac{n}{n'} C_{d,p}.$$

The loss in capacity from using systematic non-identical versions over non-systematic non-identical versions is small when $H(p)$ is small (so that $C_{d,p} \approx d(1 - H(p))$), but can be substantial when $H(p) > (1 - 1/d)$.

B. Coding Methods and Design Problems

In the case of non-identical, non-systematic versions, Raptor codes – the codes which most successfully capture the digital fountain abstraction – immediately provide a framework for achieving near-optimal performance. Each version can simply be constructed from independent symbols determined by the Raptor code. Indeed, under the setting where error rates are identical, Raptor codes appear ideal for this setting, given the recent work [10], [6].

A potential problem in this setting is that if the bits among different versions are completely independent, there is no immediate mechanism for determining the error rate for each version, which is necessary for establishing a message-passing algorithm using belief propagation. As mentioned previously, it is possible to include test bits to provide estimates. Alternatively, some derived coding bits could be used across all versions, to allow estimation methods based on comparing these values across version. This approach would be wasteful in terms of rate, but the number of bits used in all versions could be fine-tuned. The case of systematic non-identical versions can be seen as just a specific instance of this more general approach, with the corresponding advantage that a copy of the original message bits is available on every version.

We emphasize that Raptor codes are still in their infancy, and techniques for optimizing them for general error density distributions remains a challenging problem; see [10] for more details. In this setting, the design problem is further complicated by the fact that one wants a code that performs well across varying number of versions, each with a potentially different error rate, for as large a set of error rates as possible. In the systematic case, this is further complicated by the fact that the systematic bits have a different behavior than the encoding bits, which depends on the number of copies. We therefore believe this problem setting provides an interesting design challenge.

V. Conclusion

Document deterioration will continue to be a problem in the digital age, as all current storage mechanisms are subject to bit rot. As digital libraries proliferate, we expect digital document preservation to become as important a field as document preservation is today for paper-based libraries. We have considered a general problem setting where both replication and error-correction are used to provide a solution for digital deterioration, and focused on how they work together. We have explained the differences in capacity based on whether the requirements specify that version are identical and/or systematic, and highlighted the potential of recent Raptor code technology to develop practical solutions.

The theory suggests several interesting possible questions related to the design of Raptor codes. In a more practical vein, most storage media already include some limited error-correction capabilities, primarily to deal with small numbers of errors, and generally applied by logically dividing the data into small chunks rather than encoding the entire data file. It would be interesting to consider how Raptor codes or other similar codes could be used in conjunction with already existing error-correction standards.

References

[1] J. Byers, M. Luby, and M. Mitzenmacher. A Digital Fountain Approach to Asynchronous Reliable Multicast. *IEEE Journal on Selected Areas of Communications*, 20(8), pp. 1528 - 1540, Oct 2002.

[2] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, vol 18, pp. 74-93, Sept 2001.

[3] P. Maniatis, M. Roussopoulos, T.J. Giuli, D. Rosenthal, M. Baker, and Y. Muliadi. The LOCKSS Peer-to-Peer Digital Preservation System. *ACM Transactions on Computer Systems*, 23(1):2-50, 2005.

[4] M. Mitzenmacher. A note on low-density parity-check code for erasure and errors. SRC Technical Note 1998-017, COMPAQ, 1998.

[5] M. Mitzenmacher. Digital fountains: A Survey and Look Forward. In *Proceedings 2004 Information Theory workshop*, pp. 271-276.

[6] R. Palanki and J. S. Yedidia. Rateless codes on noisy channels. In *Proceedings of ISIT 2004*. Full version available as Mitsubishi Electronic Research Laboratory (MERL) TR2004-038.

[7] J. Rothenberg. Ensuring the Longevity of Digital Information, *Scientific American*, pp. 42-47, Jan 1995.

[8] T. Richardson and R. Urbanke. The Capacity of Low-Density Parity-Check Codes under Message-Passing Decoding. *IEEE Transactions on Information Theory*, 47(2), pp. 599-618, 2001.

[9] A. Shokrollahi. Raptor Codes. In *Proceedings of ISIT 2004*. (Full version to appear.)

[10] O. Etesami, M. Molkaraie, and A. Shokrollahi. Raptor codes on symmetric channels. In *Proceedings of ISIT 2004*. (Full version to appear.)

[11] I. H. Witten and D. Bainbridge. *How to Build a Digital Library*. Morgan Kaufmann, 2003.