# Robust error correction in Infofuses: Supplementary Information.

Greg Morrison, Sam W. Thomas III, Christopher N. LaFratta, Jian Guo, Manuel A. Palicios, Sameer Sonkusale, David R. Walt, George M. Whitesides, and L. Mahadevan

March 7, 2011

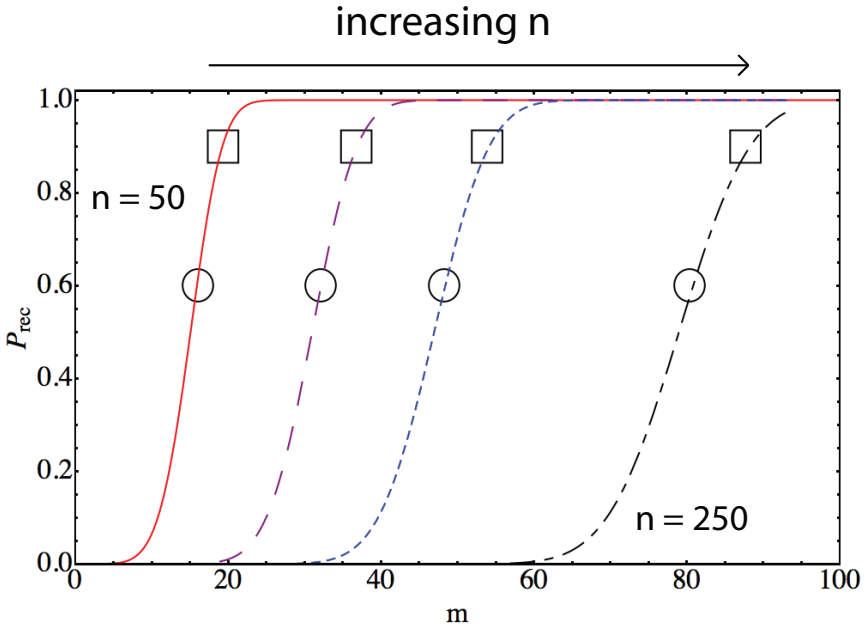


Figure 1: Predicted recovery probability using the single threshold decoding for fixed probability $p = 0.05$ as a function of $m$ for varying $n$. Solid red shows $n = 50$, dashed purple 100, dotted blue 150, and dash-dotted black 250. The large squares and circles show the predicted number of required check bits, as in Fig. 4 of the main text

# 1 Arbitrary recovery probability

An arbitrarily high probability of recovery can be demonstrated by choosing some $k_M$ such that the probability of seeing more than $k_M$ errors is $\leq \epsilon_1$ (i.e. $\sum_{k=0}^{k_M} P_{ins}(k) \geq 1 - \epsilon_1$). Further, we choose the number of check bits $m$ such that $P_{rec}(k = k_M) \geq 1 - \epsilon_2$. The probability of recovery is thus

$$\begin{aligned} P_{rec} = \sum_{k=0}^{\infty} P_{ins}(k) P_{rec}(k) &\geq (1 - \epsilon_2) \sum_{k=0}^{k_M} P_{ins}(k) \\ &\geq (1 - \epsilon_1)(1 - \epsilon_2) \end{aligned} \tag{1}$$

which can be made arbitrarily close to 1.

# 2 Optimization of the form of the check bits

The form of the check bits in eq. 1 of the main text (and likewise the form of the meta-check bits) is suboptimal, for a number of reasons. First, and perhaps most significantly, each bit at position $i \times N$ is protected *only* by the first check bit, rather than all check bits, because $iN \mathrm{mod}\ (N) = 0$. A simple improvement would be to replace the multiplying coefficient $i^{k-1}$ with $(i^*)^{k-1}$ where , $i^* - 1 = i \bmod (N-1)$. $i^*$ is chosen such that none of these coefficients is divisible by $N$. We are greatly aided by the fact that $N = 7$ is a prime; for more general $N$ we would require each of the coefficients to be relatively prime to $N$. An additional limitation of the form of the check bits in eq. 1 is that only $N$ independent check bits can have this form. The $8^{th}$ check bit will have the form $c_8 = \sum_i i^7 d_i (\mathrm{mod}\ 7) = \sum_i i d_i (\mathrm{mod}\ 7) = c_1$, since $i^7 = i\ (\mathrm{mod}\ 7)$ from Fermat's little theorem [1]. $c_7$ is highly correlated with $c_0$, differing only at the values of data points occurring at positions $i \times N$. In all of our experiments, we did not consider more than 4 check bits, so this concern is not relevant. To reliably send longer sequences (requiring more check bits), this is a significant issue, and Eq. 1 from the main text must be modified. A few examples could be $c_{k,l} = \sum_i (i^*)^k d_i^l \bmod N$ for $l \leq N-1$, or $c_{k,l} = \sum_i (i^*)^k d_i d_{i+l}$ for $l < n$. The latter is valid as long as the number of check bits is less than $n$.

For an arbitrarily long sequence, error correction in the form of a low-density parity check (LDPC) code may be optimal, where all of the data is constrained by parity checks in a sparse bipartite graph[2]. Extensive studies of LDPCs have shown them to be extremely efficient[3], coming very close to the theoretical maximum in a number of cases. LDPCs were not chosen here because they are most appropriate for long sequences, whereas communication using the infofuse is constrained by the length of the fuse. In the case of long sequences, Mitzenmacher has shown that large alphabet encoding can be exploited in the framework of an LDPC to give high-rate codes[4].

## 3 Permutation Errors

Regardless of the threshold chosen, an unexpected overlap between two peaks may give rise to a permutation error. Permutation errors in the infofuse are unique because no information is truly lost in the system. A Li peak will never be mistaken for a Cs peak, but it may be mistaken for a Li/Cs peak. A receiver who finds a pair or triple of simultaneous peaks will consider them *permutable*, in that there may have been a permutation error that produced them. In order to correct a permutation error, the receiver can generate trial sequences by splitting simultaneous peaks into their constituent parts (e. g. the Li/Cs peak would produce three trial sequences, with Li, Cs, and Li/Cs peaks) Because permutable bits may be included in the intended sequence, the probability of recovering a unique signal due to permutation errors is strongly sequence dependent.

When correcting permutation errors, the receiver generates all possible trial sequences formed by the components of each permutable peak. This gives $3^{a_0+a}7^{b_0+b}$ possible trial sequences, where $a_0$ and $b_0$ are the number of intended pairs and triples, respectively, and $a$ and $b$ are the number of pairs and triples due to permutation or insertion errors in transmission. The number of trial sequences grows very rapidly with the number of permutable bits, so that correcting permutation errors using the check bits of the form in eq. 1 in the main text rapidly becomes unweildy. It may be preferable to adapt the more computationally efficient permutation correcting codes such as Hamming[5, 6] or Reed Solomon[7] to correct for permutation errors, while retaining the presented code for correcting insertion errors.

One advantage of our error correcting code is that bursts of permutation errors can be corrected exactly. It is not difficult to see that if a single permutable bit is received (in the absence of insertion or deletion errors), only one trial sequence will reproduce the first check bit. This means our error correcting scheme can exactly correct one permutation error, as long as there are no pairs or triples in the intended sequence. If there are two permutable bits at positions $i$ and $j$, the first two check bits can be satisfied by exactly one trial sequence *unless* $i \equiv j \bmod N$. Likewise, up to 6 permutation errors can be corrected as long as the distance between permutable bits is not exactly divisible by $N$. Our error correcting scheme is most applicable for a single burst of permutation errors, where the errors occur over a relatively short range. However, as $n \to \infty$, the probability of finding a unique trial sequence becomes *independent of* $m$, the number of check bits. Thus, for long sequences, our choice of the check bits (eq. 1 from the main text) is inappropriate. A LDPC coding scheme will drastically improve the ability to correct permutation errors, but with a somewhat more complex coding and decoding scheme. In practice, we use relatively short sequences, and the probability of permutation errors tends to be relatively low. In the experiments, we initially attempt recovery under the assumption that no permutation errors have occurred, and apply a permutation-correction if the recovery fails.

In all cases where the chemicals were correctly applied, no more than 3 permutation errors were observed.

# 4 Further improvements

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | (0,2) | b | (1,3) | c | (3,0) | d | (1,5) | e | (2,0) | f | (0,0) |
| g | (1,4) | h | (2,3) | i | (0,4) | j | (6,0) | k | (3,4) | l | (5,1) |
| m | (3,1) | n | (3,2) | o | (2,1) | p | (5,0) | q | (5,3) | r | (4,0) |
| s | (0,1) | t | (1,0) | u | (2,2) | v | (0,3) | w | (1,1) | x | (3,5) |
| y | (0,5) | z | (4,1) | _ | (1,2) | . | (5,2) | ! | (4,3) | ? | (1,6) |
| , | (2,5) | ; | (4,5) | : | (2,4) | ‘ | (0,6) | ( | (3,6) | ) | (6,5) |
| 0 | (6,1) | 1 | (2,6) | 2 | (6,2) | 3 | (4,2) | 4 | (5,4) | 5 | (6,3) |
| 6 | (6,4) | 7 | (3,3) | 8 | (5,5) | 9 | (4,6) | * | (4,4) | & | (5,6) |

Table 1: Optimal alphabet minimizing the number of nearest neighbor peaks with identical chemical species. $\sim 25.1\%$ of the peaks in the sample text have 1 nearest neighbor with one element the same, $\sim 0.2\%$ have more than one element the same. Elements in the bottom two rows did not occur in the sample text, and are arbitrarily chosen. The (6,6) state is reserved as an error character.

Because we will in general include a finite number of chemical spots on an infofuse, only some of which can be check bits, we must accept that there may exist multiple trial sequences that pass the error-correction scheme, and to devise a method to choose between them. However, we are free to choose the alphabet we use (the method of mapping english characters into distinct pairings of peaks), which will allow us to select between trial sequences. We note that noise peaks are likely to share at least one chemical element in common with a nearby peak, due to the fact that noise peaks are not mis-applied chemical strips on the fuse, but rather originate from impurities in an intended peak (or the nitrocellulose itself), or due to difficult to predict fluctuations in the flame front. By choosing an alphabet such that typical sequences will have few nearest neighbor peaks with identical chemical species, we can improve our ability to distinguish between the intended sequence and spurious matches. We note that, under certain conditions, it might be more preferable to choose an alphabet that reduces permutation errors, rather than insertion errors. Such an alphabet would minimize the number of peaks that contained multiple chemical species, and would likely have many characters in common with our the insertion-minimizing alphabet.

Using Monte Carlo simulations, we have generated an alphabet that minimizes the number of peaks which share one or more elements with their nearest neighbors. We began with a random alphabet, and iteratively changed the encoding for a single character at each

MC step. The 'energy' of a particular alphabet is taken to be proportional to the total number of elements in common between nearest neighbors (different values of the proportionality constant were chosen in different runs to find fast convergence). In order to model correlations in the English language, we chose as a sample text the book of Genesis from the King James bible [8] (which allows us to account for the correlations between letters in English). The resulting alphabet is shown in Table 1, and we find $\sim 25\%$ of nearest neighbor peaks share identical elements. This can be compared to our original alphabet (generated in an ad-hoc fashion), with $\sim 70\%$ of the nearest neighbors sharing identical elements. This alphabet has the additional advantage that peak pairs or triplets are more rare, thus reducing the number of permutable bits in an intended sequence. Incorrect trial sequences that *do* pass the check bits are likely to differ significantly from the intended sequence as long as the number of check bits is sufficiently large, so that incorrect sequences are expected to have a large number of peaks with identical elements.

In practice, we will send short messages using the infofuse, rather than an entire book. It is then important to determine whether or not developing the alphabet in this way will give good results for shorter sequences. We simulate the transmission of a block of 7 characters (14 bits) and 2-4 check bits with an insertion probability of $p = 0.1$ 500 times, and determine the number of trial sequences that have more nearest neighbor peaks with identical species. If we take inserted peaks to have a random data value, we find $\sim 20\%$ of the trial sequences that satisfy the check bits have the same number of nearest neighbor peaks composed of the same chemicals and $\sim 70\%$ have more, regardless of the number of checkbits. If we assume that inserted peaks are correlated, such that the inserted data value matches one of its nearest neighbors, we find $\sim 80\%$ of the accepted trial sequences have more nearest neighbors with the same chemicals.

## References


[1] J. Stillwell. *Elements of Number Theory.* Springer-Verlag, New York, 2003.

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

[3] D. J. C. MacKay. *IEEE Trans. Info. Theory*, 45:399, 1999.

[4] M. Mitzenmacher. *IEEE Trans. Info. Theory*, 52:5496, 2006.

[5] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley, New York, 1991.

[6] R. W. Hamming. *Bell System Tech. J.*, 29:147, 1950.

[7] S. B. Wicker and V. K. Bhargava. *Reed-Solomon Codes and their applications.* IEEE Press, Piscataway, NJ, 1994.

[8] Project Gutenberg. *http://www.gutenberg.org/etext/8001*.