Corruption Resilient Fountain Codes
Technical Report #2008-12
July 2008

Shlomi Dolev Nir Tzachar

August 5, 2008

Abstract

A new aspect for erasure coding is considered, namely, the possibility that some portion of the arriving packets are corrupted in an undetectable fashion. In practice, the corrupted packets may be attributed to a portion of the communication paths that are leading to the receiver and are controlled by an adversary. Alternatively, in case packets are collected from several sources, the corruption may be attributed to a portion of the sources that are malicious.

Corruption resistant fountain codes are presented; the codes resemble and extend the LT and Raptor codes. To overcome the corrupted packets received, our codes use information theoretics techniques, rather than cryptographic primitives such as homomorphic one-way-(hash) functions. Our schemes overcome adversaries by means of using slightly more packets than the minimal number required for revealing the encoded message, and using a majority over the possible decoded values. We also present a more efficient randomized decoding scheme.

Beyond the obvious use, as a rateless erasure code, our code has several important applications in the realm of distributed computing.

1 Introduction

Modern erasure codes allow efficient encoding schemes for use in a wide range of applications. We suggest to augment erasure codes with a capability to withstand malicious packet corruption. Next, we list several core applications and explain our contribution in each case.

Erasure codes. Recent advances in erasure coding have produced *digital fountain* codes [2], for example, Raptor codes [16] and LT codes [9], which are (practically) infinite rate erasure codes. These codes allow linear time encoding and decoding, with high probability. Alas, to the best of our knowledge, such codes are only resilient to lossy channels. Moreover, when introducing simple fault models, such as models that allow bit corruption, the only solution offered is pipelining the existing code's packets with an error correcting code. As a consequence, such solutions cannot cope with *Byzantine* channels, where an adversary may arbitrarily change packets; for instance, when communication is done over parallel paths and some paths are under the control of an adversary.

Consider, for example, a Raptor code, where encoding packets are sent in order to transfer a message. Each packet sent is further encoded by using an error correcting code. Consider an adversary which corrupts a single packet and its error correction padding, such that the error correcting code does not identify the packet as a corrupted packet. This corrupted packet may well prevent the receiver from correctly decoding the original message that should have been transfered.

In this work, we present a digital fountain code which resembles LT codes. Our code is resilient to Byzantine channels. Our code is information theoretic secure.

Shared value. Beyond the obvious uses as a classical error correcting code, the new scheme can be used to maintain a distributed *shared value*, where, for example, a group of sensors receive and record global inputs. The inputs may be from a control and command entity, such as a satellite, or a natural event that the sensors sense, such as an earthquake. The sensors wish to store the input for later retrieval. In other cases, the sensors may need to store a global history for later retrieval.

When maintaining copies of data is too expensive, one would prefer a scheme based on error-correcting codes which reduces storage requirements at each individual sensor while still able to cope with faulty sensors, which may hold corrupted information.

We present a randomized scheme in which shared data is efficiently recorded with *no communication* among the sensor. Note that, in some cases, it is also possible to update the encoded data without decoding, as suggested in [7].

Erasure coding with augmented Byzantine indication. Several works present different strategies of coping with Byzantine adversaries, both in erasure coding and network coding scenarios. A common approach to overcome Byzantine adversaries when implementing erasure codes is checking each received packet against a pre-computed hash value, to verify the integrity of the packet. When using fixed rate codes, the sender can pre-compute the hash value of each possible packet, and publish this hash collection in a secure location. The receiver first retrieves this pre-computed hash collection, and verifies each packet against the packet's hash as the packet arrives. The hash is a one way function and, therefore, when the adversary is computationally limited, the adversary cannot introduce another packet with the same hash. However, when using rateless codes, such techniques are not feasible; as there is practically an infinite number of different packets, there is no efficient way to pre-compute the hash value of each possible packet and send these hashes to the receiver.

The authors of [10] implement a slightly different technique for packet verification in rateless codes. The technique proposes the use of a Merkle-tree [14] based signature structure. However, the solution proposed is, still, only valid against computationally bound adversaries, and relies on the existence of homomorphic, collision-resistant hash functions. Furthermore, as the size of a Merkle-tree is linear in the size of the original message, the authors propose a process of repeated hashing to reduce the size of the tree. Such recursive application of a hashing function is more likely to be susceptible to attack. Another problem inherent to hashing technique is secure publication of the hashes. The sender must devise a way to securely transfer hashes to the receiver, say, by an expensive flooding technique.

In contrast, we provide the first information theoretically secure rateless erasure code.

Network coding with auxiliary data. Although not immediately apparent, network codes are closely related to rateless erasure codes. Erasure codes may benefit from techniques to cope with Byzantine adversaries developed for network coding, and vice versa. Several network coding related papers discuss the merits of using hash functions to overcome Byzantine adversaries in network coding protocols, all of which require out of band communication or preprocessing. Other protocols employ some kind of shared secret between the sender and receiver to cope with computationally bounded adversaries.

A different solution appears in [17], in which the only assumption needed to overcome an all-powerful adversary is a shared value between the sender and the receiver, which may also be known to the adversary (in [17], this shared value is termed the redundancy matrix). Using this

shared value, and sending enough redundant information, the receiver can overcome a Byzantine adversary, as long as the adversary cannot control more than half of the network's capacity (which is the minimal cut between the sender and receiver). However, this solution requires, yet again, out of band communication or preprocessing, as the sender and receiver must share a value before starting the communication. Furthermore, it is not straightforward to implement such a technique in erasure coding systems, in particular when trying to receive a message by employing several sources in parallel, as the solution presented in [17] requires batching packets into groups of predetermined size.

In [11], Koetter and Kschischang present a different, sophisticated, approach, based on high dimensional vector spaces; a message of $m \cdot k$ bits is encoded into a vector space, V, of dimension $l \leq m$, which is a subspace of an ambient vector space W of dimension $l + m$. l is a parameter of the encoding scheme, m is the number of bits in a message block and k is the number of blocks in the message. Each packet the sender creates is a randomly chosen vector (of $l + m$ bits) in V. The receiver, upon collecting enough vectors – l linearly independent vectors – can proceed to reconstruct the original message from the received vector space U. The authors present a minimal distance decoder, which can recover V from U provided that, when writing U as $U = V \cap U + E$ where E is the error space, $t = dim(E)$ and $\rho = dim(V \cap U)$, it holds that $\rho + t < l - k + 1$.

The codes presented in [11] have theoretical contribution, however, they may suffer from several sever implementation problems; to boost the error resiliency of the code, one should (a) increase l or (b) decrease k. The implications are sending more redundant information in each packet (increasing l) and having larger packet sizes (decreasing k). Moreover, to recover from a Byzantine attack on at most a third of the packets sent, the codes presented require that $m \in \Omega(\sqrt{n})$ (that is, each block must be of length $\Omega(\sqrt{n})$ bits), where n is the size of the message. In contrast, our codes are not limited by block sizes, and can cope with a third of Byzantine corrupted packets regardless of the block size.

Paper organization. The system settings and attack strategies on existing codes appear in Section 2. Our new local codes appear in Section 3. Applications are described in Section 4. The paper is concluded in Section 5. Detailed descriptions of the application are omitted.

2 System Settings and Attack Strategies

A rateless erasure code is defined by a pair of algorithms, $(\mathcal{E}, \mathcal{D})$, such that, given a set of input symbols, $\mathcal{E}$ can produce a practically infinite sequence of different output *packets*. Moreover, from any large enough set of packets, $\mathcal{D}$ can be used to recover all of the original symbols.

A rateless erasure code is usually used between a *sender* and a *receiver*, where the sender wishes to send a specific message to the receiver. The sender starts by dividing the message into symbols, and then uses $\mathcal{E}$ to generate packets, which are then sent to the receiver over a lossy channel. The receiver, after collecting enough packets, uses $\mathcal{D}$ to recover the original symbols, and, from them, the message.

Next, we define the adversarial model we use. We assume that the computation power of the adversary is unlimited, and the adversary may sniff all traffic in the network. Furthermore, the adversary may forge or alter packets such that the receiver cannot differentiate them from legitimate packets. The only restriction we place on the adversary is the number of packets the adversary may corrupt. The restriction is defined by looking at all packets arriving at the receiver, according to their arrival time (receiver subjective time). We then say that an adversary is c-bounded, with

a parameter $c \leq \frac{1}{3}$, if, for each $i \geq 4$ and for each sequence of length $\geq i$ of consecutive packets, no more than $c \cdot i$ packets are corrupted.

Given the above settings, we discuss the possible ways an adversary may influence the Belief Propagation decoding algorithm, used by [9, 16]. The Belief Propagation decoding algorithm suits the following succinct encoding algorithm: to generate a packet, choose a random subset of input symbols and XOR them. The choice of the random set of neighbors forms the critical part of the encoding algorithm, and defines the number of packets needed for correctly decoding the input symbols. For an in-depth discussion, see [9].

The Belief Propagation decoding algorithm works as follows: given a set of packets, define a bipartite graph, $G = (A, B, E)$, where the bottom layer, A, contains the packets and the upper layer, B, the input symbols. An edge exists between a packet $a \in A$ and a symbol $b \in B$, if b was used in generating a. The Belief Propagation decoder is described in Figure 1, where the successful completion of the decoding process depends on the neighbor distribution.

Next we show the vulnerability of the Belief Propagation decoder. Note that our tests show that the specific attacks we list and prove, are not the most severe. Our tests show that corrupting a very small portion of the packets corrupts almost half of the symbols.

definitions:
$N(a) = \{b \in A \cup B | (a, b) \in E\}$

While$(\exists b \in B : |N(b)| > 0)$
  Let p be a packet, such that $N(p) = \{s\}, s \in B$.
  Copy p to its only neighbor, s, which is then successfully decoded.
  Foreach $a \in N(s)$ **do**
    $a \leftarrow a \oplus s$
    $E \leftarrow E \setminus \{(a, s)\}$.
  Done
Done

Figure 1: Belief Propagation decoder

Attacking the Belief Propagation decoding algorithm. In the simple scenario depicted in Figure 2, the adversary can corrupt each decoded symbol by altering a single encoding packet overall. In general, it would be interesting to find the best possible strategy for the offline and online adversaries.

• Offline attack. Assuming the adversary knows, in advance, the graph generated at the **receiver**, what would be the optimal strategy for corrupting the largest number of decoded symbols, given that the adversary is c-bounded, for some constant c. This immediately translates to an upper bound on the number of symbols the adversary can corrupt, a bound which may be employed in

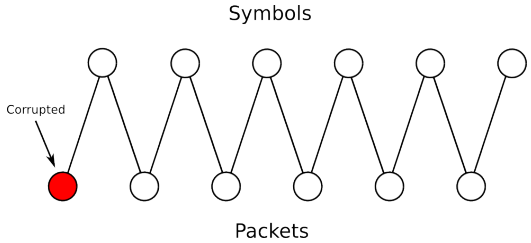

Figure 2: Simple Attack scenario

devising techniques to overcome the adversary.
• Online attack. A different scenario is in the case in which the adversary does not know the bipartite graph in advance, and needs to choose which packets to alter/inject into the system, as they traverse the communication system.
• Odd packets attack. A simple attack, which corrupts all decoded symbols, may be devised for both the online and offline attack scenarios. Consider corrupting all packets which have an odd degree, i.e., connected to an odd number of symbols. Corrupt each packet by flipping all bits (or a subset thereof). Using a simple inductive argument, we are able to prove that the resulting decoded symbols from the Belief Propagation decoder will all be flipped.

We now compute the number of packets that have an odd number of neighbors. We will show that when using the Robust Soliton distribution from [9], the expected number of such odd degree packets is less than one third of the number of packets.

We start by analyzing the Ideal Soliton distribution from [9], which specifies that, for k input symbols, the degree distribution of each encoded packet is:

$$P[degree = i] = \rho(i) = \begin{cases} \frac{1}{k} & i = 1 \\ \frac{1}{i(i-1)} & i \geq 2 \end{cases}$$

The probability that, for a given packet p, the degree is odd:

$$\begin{aligned} P[degree(p) \text{ is odd}] = \sum_{i \geq 1, i \text{ is odd}}^{k} \rho(i) &\leq \frac{1}{k} + \frac{1}{3 \cdot 2} + \frac{1}{5 \cdot 4} + \frac{1}{7 \cdot 6} + \cdots \\ &= \frac{1}{k} + \sum_{i=2}^{\infty} \frac{(-1)^i}{i} = \frac{1}{k} + 1 - ln(2) \end{aligned}$$

We get that for $k \geq 38$, the probability for each packet to be of odd degree is less than $\frac{1}{3}$. Now, consider a binomial random variable, $X \sim B(n, p)$, such that n equals the number of packets and $1/6 \leq p \leq 1/3$. The adversary can successfully corrupt all packets, as long as $X \leq n/3$. Using the normal approximation for the binomial distribution (where $Z \sim U(0, 1)$), we get that:

$$P[X \leq n/3] \approx P[Z \leq \frac{n/3 - np}{\sqrt{np(1-p)}}] \geq \frac{1}{2}$$

Therefore, the adversary has a probability of at least half to corrupt all decoded symbols by corrupting at most one third of the packets. Moreover, when using the Robust Soliton distribution from [9], one may show that the probability of each packet being of odd degree is, again, less than one third, which implies that, even when using the Robust Soliton distribution, the adversary still has a probability of at least one half to corrupt all decoded symbols.

3 Corruption Resilient Fountain Codes

3.1 Encoding

To encode a given message of n bits, split the message into m pieces (the input symbols), $k_0, k_1, \ldots k_{m-1}$, each of length $k = \frac{n}{m}$ bits. For each $0 \leq j < m$, let k_i^j be the j'th bit of the i'th message piece.

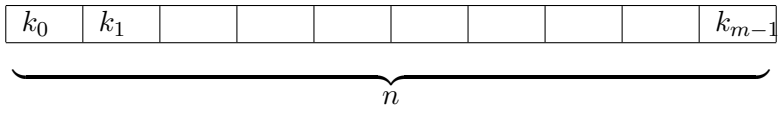

Figure 3: Splitting a message

For a k dimensional vector $\overline{v} = (v_0, v_1, \ldots, v_{k-1})$ in $\mathbb{GF}^k(2)$ define the characteristic boolean function $f_{\overline{v}} : \mathbb{GF}^k(2) \to \mathbb{GF}(2)$ in the following way: $f_{\overline{v}}(x_0, x_1, \ldots, x_{k-1}) = \bigoplus_j x_j v_j$ (in other words, the inner product of x and v over $\mathbb{GF}^k(2)$).

To generate a packet p, randomly select a k dimensional vector $\overline{r} \in \mathbb{GF}^k(2)$ (the distribution used to sample $\overline{r}$ will be discussed later), and set $p = \langle r, f_{\overline{r}}(k_0), f_{\overline{r}}(k_1), \ldots, f_{\overline{r}}(k_{m-1})\rangle$. Note that, for brevity, each packet is assumed to be of length $k + m$ bits; later, we show how to reduce the amount of redundancy needed from k to $\log^2 k$ bits, in essence compressing the r vector. We define the following:

Definition 3.1. Two packets, p_1 and p_2, are termed independent if their associated $\bar{r}$ vectors are independent over $\mathbb{GF}^k(2)$.

Alternatively, a more efficient decoding (hardware-wise) is achieved by setting $v_i = \langle k_0^i, k_1^i, \ldots, k_{m-1}^i\rangle$ and then $p = \langle r, f_{\overline{r}}(v_0), f_{\overline{r}}(v_1), f_{\overline{r}}(v_{m-1})\rangle$. This representation is more efficient as it is faster to XOR entire *words* than individual *bits*. We note that such encoding is similar to the one presented in [9]. We will later also employ the Robust Soliton Distribution presented therein in our calculations.

The decoding procedures below are applicable for both encoding alternatives. For brevity, we only discuss the first encoding alternative.

3.2 Decoding

We present several possible ways to decode a value, where the trade off between the number of packets which need to be collected and the decoding time is investigated.

We will limit the discussion to decoding a given message piece, k_l, where all message pieces may be decoded in parallel, using the same technique.

Majority voting. The decoding algorithm is depicted in Figure 4. To reconstruct a given message piece, k_l, given that at most f faults occurred, we need to collect $2f + 1$ pairwise disjoint sets of packets, such that each set contains exactly k independent packets. From each independent set, S_j, we can reconstruct an s_j as a candidate message piece. It then follows that the majority of the values is the correct message piece. In other words,

$$k_l = \operatorname*{argmax}_{s_j} |\{s_i : s_i = s_j\}|$$

Majority voting decoding may only be used when the adversary is at most $\frac{1}{2k}$-bounded, to ensure that p, the number of packets collected, will suffice to compose $(2pc + 1)$ sets of k packets.

Exhaustive search algorithm. For both this decoding algorithm and the next one, we will need the following lemma:

Lemma 3.1. *Let $N = \{p_j | p_j = (r_j, f_{\overline{k_l}}(r_j))\}$ be a set of packets, such that $|N| \geq k+2f+\epsilon$. Define the following matrix, $\hat{A} = (r_j)$ and let $\bar{b} = (f_{\overline{k_l}}(r_j))$. Assuming that no more than $f < k$ packets*

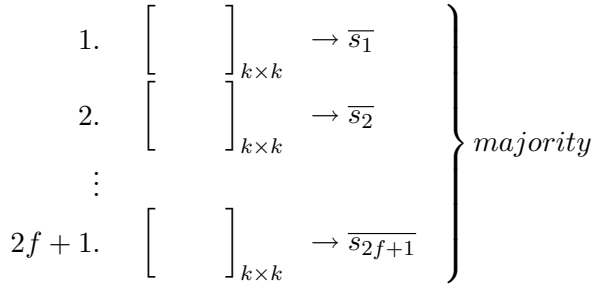

Figure 4: Using majority logic to decode a message piece

are corrupted, and that with very high probability (for a specific ϵ, to be defined later) $k+\epsilon$ packets contain a subset of size k of independent packets, then k_l is the only solution to the following equation which satisfies at least $k+f+\epsilon$ equations (where each row in $\hat{A}$ is viewed as a single equation):

$$\hat{A} \cdot \bar{x} = \bar{b}$$

Proof. The proof follows from the fact that each solution satisfying at least $k+f+\epsilon$ equations must satisfy at least $k+\epsilon$ uncorrupted equations. Furthermore, with very high probability, these $k+\epsilon$ uncorrupted equations contain a subset of k independent equations, and as there is a only single solution to any system of k independent, uncorrupted equations, the proof concludes. □

It follows from Lemma 3.1 that the solution, $\bar{x}$, to the system of equations $\hat{A} \cdot \bar{x} = \bar{b}$, which correctly solves the largest number of equations is the (only) correct solution. Hence, an algorithm which solves the following optimization problem would have been invaluable: given a system of equations over $\mathbb{GF}(2)$, find the best possible $\bar{x}$, which solves the maximal number of equations. Obviously, this is an NP-complete problem (by a simple reduction from the max-cut problem, see [1, 8]).

Given Lemma 3.1 and the above argument, a simple exhaustive search over all possible k_l values yields the correct answer, which satisfies at least $k+f+\epsilon$ equations out of N. Exhaustive search decoding is applicable to all c-bounded adversaries, where $c < \frac{1}{3}$. Nevertheless, as such a search is exponential in k, it may not be applicable in all situations. Next, we present better decoding algorithms, which trade decoding time for increasing amounts of packets, assuming that the adversary is bounded by smaller values than 1/3.

Randomized decoding algorithm. We use randomization in order to reduce the decoding complexity, given that the adversary is c-bounded (for an appropriate c, to be defined later), and may only corrupt at most f packets. We will follow Lemma 3.1 from the previous section. The algorithm is depicted in Figure 5. Assume that we have collected N packets, such that $|N| = g(k)+k+f+\epsilon \geq k+2f+\epsilon$ for some $g(x) \geq f$ to be defined later. Each subset of $k+\epsilon$ uncorrupted packets has a very high probability (again, as a function of ϵ) of containing a subset of k independent packets. Let this probability be p_ϵ. The algorithm will work as follows: choose a random subset, $S \subset N$, such that $|S| = k+\epsilon$. Check if there exists a subset $S' \subset S$ which contains exactly k independent packets. If such an S' exists, let $S' = \{c_j\}$ and define $\hat{A} = (r_j)$ and $\bar{b} = (f_{\overline{k_l}}(r_j))$ and let $\bar{s}$ be the solution to the equation $\hat{A} \cdot \bar{x} = \bar{b}$.

If the obtained solution, $\bar{s}$, satisfies more than $k+f+\epsilon$ equations out of N, then, according to Lemma 3.1, $k_l = s$ and we are done.

N : A set of $g(k) + k + f + \epsilon$ packets

Predicates:
candidate(S) := S contains exactly k independent packets
good$(\bar{s})$:= $\bar{s}$ satisfies at least $k + f + \epsilon$ equations out of N

Repeat
1 $S \leftarrow$ a random set of N, $|S| = k + e$
2 *if* $\exists S' \subset S \wedge$ *candidate*(s) **then**
3 Let $S' = \{c_j\}$
4 $\hat{A} \leftarrow (r_j)$
5 $\bar{b} \leftarrow (f_{\overline{kl}}(r_j))$
6 solve $\hat{A} \cdot \bar{x} = \bar{b}$
7 $\bar{s} \leftarrow \bar{x}$
8 **if** *good*$(\bar{s})$ **then**
9 **return** s
10 **fi**
11 *fi*
Done

Figure 5: Randomized decoding algorithm

Now, let p_k be the probability of choosing a subset of N with no corrupted packets. It then follows that the expected runtime of the algorithm is

$$\frac{1}{p_k \cdot p_\epsilon}$$

Let us now approximate p_k:

$$\begin{aligned} p_k &= \frac{\binom{g(k)+k+\epsilon}{k+\epsilon}}{\binom{g(k)+k+f+\epsilon}{k+\epsilon}} = \frac{\frac{(g(k)+k+\epsilon)!}{g(k)!}}{\frac{(g(k)+k+f+\epsilon)!}{(g(k)+f)!}} \\ &= \frac{(g(k)+1)(g(k)+2)\cdots(g(k)+f)}{(g(k)+k+1+\epsilon)(g(k)+k+2+\epsilon)\cdots(g(k)+k+f+\epsilon)} \\ &\geq \left(\frac{g(k)}{g(k)+k+\epsilon}\right)^f = \left(1+\frac{k+\epsilon}{g(k)}\right)^{-f} > \exp\left(-f\frac{k+\epsilon}{g(k)}\right) \end{aligned}$$

Assuming that p_ϵ is a constant close to 1, we can choose $g(k)$ according to our needs – either minimizing decoding time or minimizing the number of packets we need to collect. For example, choosing $g(x) > \frac{f\cdot(k+\epsilon)}{\log b}$, for a constant b, results with $p_k > 1/b$. Such a choice minimizes the run time, at the expense of having to collect many messages ($g(x) \approx kf$). Furthermore, such a decoding algorithm is only relevant when the adversary is at most $\frac{1}{k}$-bounded.

Overall, the complexity of decoding the entire message is in $O(n + m \cdot k^3) = O(n + \frac{n^3}{m^2})$.

Calculating ϵ. The choice of ϵ, that should ensure each $k + \epsilon$ packets are independent with probability p_ϵ, is entirely dependent on the distribution from which we draw the random vectors during the encoding stage.

In [9] the author presents a distribution, termed the Robust Soliton distribution, which ensures that if there are no faults, the correct message can be decoded by using $k + O(\sqrt{k}\log^2(k/\delta))$ packets with probability at least $1 - \delta$ for any δ, using Belief Propagation decoding (see [9] for

specific details). When using the Robust Soliton distribution, we can set $\epsilon = O(\sqrt{k}\log^2(k/\delta))$ and then $p_\epsilon = 1 - \delta$.

However, as we have to solve a system of equations (e.g., by using Gaussian elimination), the use of the Robust Soliton distribution is negated, as this distribution was designed to facilitate Belief Propagation decoding. In our coding scheme, we employ a random choice of neighbors for each encoded symbol, resulting in a system of m equations over k variables. Each coefficient is then a random variable in $\mathbb{GF}^k(2)$. Such a system equations has a solution as long as the system is of full rank, k. In [16] the author simply demonstrates that the probability that the system is not of full rank is at most 2^{k-m}. Choosing $m = k + c\log k$ shows that the system is of full rank with a probability larger than $1 - 1/k^c$. Therefore, setting $\epsilon = c\log k$ suffices, in this case.

Reducing packets size. In Section 3.1, we assume that each packet generated is of size $k + m$ bits, where m is the size of each message block and k is the number of blocks. The k bits of redundancy are used to denote which blocks participated in the creation of the packet. In practice, these k bits may be compressed, using several techniques, into a logarithmic factor. First, when choosing which blocks to XOR in a uniform fashion, the sender can use a PRNG to generate the required distribution, as proposed in [9]. The sender will use a different random seed for the PRNG for every packet generated, and only attach the seed to the packet. A seed of size logarithmic in k suffices, and the receiver can proceed to recover which blocks participated in creating a specific packet using the seed embedded in the packet.

A different approach, in which the use of a PRNG is not required, can be achieved by using a different distribution than the uniform distribution to select blocks for each packet; first generate a binary vector, r, of length k, in which each index is 1 with probability $\frac{(1+\delta)\log k}{k}$, for some small constant δ. When sending each packet, do not attach r to the packet. Rather, attach the indices of r which contain 1. On average, each packet carries $\log^2 k$ redundant bits. Next we use [4] to define the appropriate values for ϵ and p_ϵ. Theorem 1 of [4] (adapted to our notations) states that given a matrix A over $\mathbb{GF}(2)$, of dimension $(k+\epsilon) \times k$, with 1's chosen according to the distribution above, the probability that A is non singular is:

$$\lim_{k\to\infty} P[A \text{ is not singular}] = \prod_{j=\epsilon+1}^{\infty} \left(1 - 2^{-j}\right)$$

Moreover, it can be verified that the expression above holds even for values of k as small as 10. Choosing a constant value for ϵ results, with very high probability, with a non singular matrix. This further implies that a set of $k + \epsilon$ vectors, chosen using the distribution discussed above, contains a subset of k independent vector.

4 Applications

Erasure coding. Consider a Peer-to-Peer system, where a user would like to receive some content. To alleviate the load on any single Peer, the content may be mirrored at several Peers. On the other hand, to maximize bandwidth usage, the user should be able to receive parts of the content from several mirrors in parallel. Erasure codes, and in particular digital fountain codes, give rise to a simple solution to this problem; each mirror locally, and independently, generates packets and sends them to the user. If a total of k packets are needed to reconstruct the content, and there are m mirrors, each mirror need only send $O(\frac{k}{m})$ packets.

Alas, the system described above is very sensitive to Byzantine Peers; when even one of the mirrors intentionally corrupts packets, the receiver will never be able to reconstruct the requested content. A more robust solution is to use local codes, presented in the previous section, such that a constant fraction of Byzantine mirrors can be tolerated.

Corollary 4.1. *Local codes solve the erasure coding problem with Byzantine adversaries.*

Shared value. Consider a given set of nodes, which initially share a value x of k bits. We wish to reduce the storage requirements at each node, such that each node will only need to store a fraction of k bits of the original value. We require that *no communication* takes place during the initial stage, so that each node generates its own encoded (short) share of x independently of other nodes. The nodes may communicate later to reconstruct x from the stored shares. Moreover, the solution should be robust against a constant fraction of Byzantine nodes, where a Byzantine node may introduce arbitrary information into the system.

Our local code can be used to solve the *shared value* problem; Assuming that all nodes initially receive/sense the value, x, each node generates a packet for x according to the local code scheme presented in Section 3, using a predetermined value for k. When a node, p, needs to reconstruct the original value, p collects packets from enough nodes, such that x may be decoded correctly.

Assume that a fraction c of *all* the packets in the system are corrupted; p will then collect a set of packets, N, by selecting nodes to communicate with at random. p will then use one of the decoding algorithms presented above, which suit a c-bounded adversary, to recover x.

A different scenario to consider is a sensor network, where a node may reliably communicate only with the node's neighbors. As there may not be a direct radio link between all nodes, some routing protocol must be employed. To cope with Byzantine neighbors, which may alter packets passing through them, k can be chosen such that each node has enough neighbors to recover x, given that at most a fraction of c of the neighbors are corrupted.

Corollary 4.2. *Local codes solve the shared value problem.*

A different approach to the one taken above can be achieved by using error correcting codes (for example, A Reed-Solomon code), when nodes have, or can randomly choose, distinct identifiers. Namely, encoding a shared value x at each node by using a $[k, k-2f]$ code in the following way: let $k = 3f + 1$. Each node builds a polynomial g from x, of degree at most $k - 2f$ (by padding x with zeroes as necessary), in a deterministic way, such that each node builds the same g. Finally, each node p evaluates g at a point which corresponds to p's identifier, e.g., $g(id_p)$, stores the result and discards x.

When a node, p, wishes to reconstruct x, p needs to collect k distinct values from other sensors (and p), which, together with the value stored locally at p, can be used to reconstruct x, overcoming at most f faulty values (possibly caused by Byzantine nodes).

5 Concluding Remarks

Error correcting codes, erasure correcting codes, communication networks and distributed computing are tightly coupled. The replication techniques used for obtaining fault-tolerance in distributed computing may be replaced by error and erasure correcting techniques. Beyond the memory overhead benefit, the dispersal of information can be useful to protect and hide clear-text values when necessary. In this work, we have presented efficient encoding and decoding schemes that enhance

the correctability of well known rateless erasure correcting codes.

Acknowledgments. It is a pleasure to thank Michael Mitzenmacher for helpful inputs and for pointing out relevant related works.

References

[1] Edoardo Amaldi, Viggo Kann, "The complexity and approximability of finding maximum feasible subsystems of linear relations", *Theoretical Computer Science*, Volume 147, Issues 1-2, August 1995, pp. 181–210.

[2] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, "A Digital Fountain Approach to the Reliable Distribution of Bulk Data", *ACM SIGCOMM 1998*, pp. 56–67.

[3] Amos Beimel, Shlomi Dolev, and Noam Singer, "RT oblivious erasure correcting", *IEEE/ACM Transactions on Networking*, Volume 15 No. 6, pp. 1321-1332, December 2007.

[4] Colin Cooper, "On the distribution of rank of a random matrix over a finite field", *Random Structures and Algorithms*, Volume 17, Issue 3-4, pp. 197–212.

[5] Shlomi Dolev, Ted Herman, "Parallel composition for time-to-fault adaptive stabilization", *Distributed Computing* Volume 20(1), pp. 29–38, 2007.

[6] Shlomi Dolev, Boris Fitingof, Avraham Melkman, and Olga Tubman, "Smooth and Adaptive Forward Erasure Correcting". *Computer Networks Journal*, special edition on Overlay Networks, Volume 36, Issue 2-3, (July 2001) pp. 343–355.

[7] Shlomi Dolev, Limor Lahiani, and Moti Yung, "Secret Swarm Unit, Reactive k-Secret Sharing", INDOCRYPT, pp. 123–137, 2007.

[8] Johan Håstad, "Some optimal inapproximability results", *Journal of the ACM*, Volume 48, Issue 4, 2001, pp. 798–859.

[9] Michael Luby, "LT Codes", *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 271–280, 2002.

[10] Maxwell N. Krohn, Michael J. Freedman and David Mazières, "On-the-fly verification of rateless erasure codes for efficient content distribution", pp. 226–240 *Proceedings of the IEEE Symposium on Security and Privacy, 2004*.

[11] Ralf Koetter, Frank Kschischang, "Coding for Errors and Erasures in Random Network Coding", `http://aps.arxiv.org/pdf/cs.IT/0703061`.

[12] Shay Kutten, Boaz Patt-Shamir, "Time-adaptive self stabilization", *Annual ACM Symposium on Principles of Distributed Computing*, pp. 149–158 ,1997.

[13] Shay Kutten, Boaz Patt-Shamir, "Asynchronous time-adaptive self stabilization", *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 149–158 , 1998.

[14] Ralph C. Merkle, "A digital signature based on a conventional encryption function", Crypto, pp. 369–378, 1987.

[15] Michael Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance",*Journal of the ACM*, 36, 2 (Apr. 1989), pp. 335–348.

[16] Amin Shokrollahi, "Raptor Codes", *IEEE Transactions on Information Theory*, Volume 52, pp. 2551–2567, 2006.

[17] Sidharth Jaggi, Michael Langberg, Sachin Katti, Tracy Ho, Dina Katabi, Muriel Medard, "Resilient Network Coding in the Presence of Byzantine Adversaries", INFOCOM 2007: *26th IEEE International Conference on Computer Communications*, pp. 616–624, May 2007.