DISTRIBUTED FOUNTAIN CODES FOR NETWORKED STORAGE

Alexandros G. Dimakis, Vinod Prabhakaran , Kannan Ramchandran

Department of Electrical Engineering and Computer Science,
University of California, Berkeley, CA 94704.
Email: {adim, vinodmp, kannanr}@eecs.berkeley.edu

ABSTRACT

We investigate the problem of constructing fountain codes for distributed storage in sensor networks. Specifically, we assume that there are n storage nodes with limited memory and $k < n$ data nodes generating the data by sensing the environment. We want a data collector who can appear anywhere in the network, to query *any* $k + \epsilon$ *storage nodes* and be able to retrieve *almost all the data packets*. We demonstrate how it is possible to solve this problem by using a specific kind of fountain code that requires only linear communication and decoding complexity. Further, for a grid topology, we propose a randomized algorithm that constructs the fountain code over a network using only geographical knowledge and local decisions. A key step in the analysis of our algorithm is a novel result concerning random walks on finite grids with traps.

1. INTRODUCTION

In this paper we address the problem of creating a robust, distributed network memory hence providing fast and reliable access to distributed data using unreliable sensor nodes. The popular approach to retrieving data in wireless sensor networks is for the data collector to query for the data from the sensor nodes of interest. The desired data is then routed from the source nodes to the data collector. This may be categorized as a "pull-based" strategy. In certain scenarios of interest, a pull-based approach at query time may have limitations. Primarily, there can potentially be a large latency in getting the desired data out of a multitude of source nodes scattered randomly across the network due to the multi-hop routing phase following the query. In addition, storage nodes may fail and redundancy might be necessary to ensure that important data will be preserved. There is a tradeoff between the work performed at the time that the data is generated relative to the work performed at query time. In general, processing done at query time introduces latency and unreliability that may not be acceptable for certain applications. This work is accordingly motivated at trying to reduce latency and unreliability between query time and the time that the desired data is made available to the data collector.

Motivated by "smart dust" sensor networks [9], we consider a large scale network with unreliable nodes that have constrained communication, computation, and storage capabilities. Given k data nodes sensing some physical quantity of interest, we would like to use n unreliable storage nodes as a robust distributed network memory. The key issue is to introduce redundancy for reliability while at the same time minimize the required communication cost.

This research was supported by NSF under grants CCR-0219722 and CCR-0330514.

1.1. Problem Description

We assume that there are k data-generating nodes that are measuring a physical quantity of interest. Without loss of generality, we will assume that each data node generates one data packet of significant size containing the measurements over some time interval.

In our problem setting, we will assume that the data packets are *independent*. In most interesting sensing scenarios the data will be highly correlated and our scheme can be combined with distributed source coding to compress correlated data. Essentially, after distributed compression, the large correlated data packets can be replaced by smaller packets that are independent and have the theoretically smallest possible size, equal to the joint entropy of the physical sources.

Further, assume we have $n > k$ *storage* nodes that will act as storage and relay devices. Sensor nodes have limited memory and we model that by assuming that each storage node can store only one data packet (or a combination having the same number of bits as a data packet). This is required for the scalability of the network.

The ratio $k/n < 1$ is assumed fixed as k and n scale. For example we can assume that some fixed ratio (for example 10%) of nodes in a sensor network are measuring while the rest are used as storage and relay devices.

We want to add redundancy and store the information contained in the k data packets in the n storage nodes. As there are k data packets of interest, and each storage node can store no more than 1 data packets worth of bits, it is clear that one has to query at least k storage nodes to completely recover the original data. We will refer to this procedure of storing the k data packets in n storage nodes and adding redundancy as a *distributed networked storage strategy*.

1.2. Distributed Networked Storage

The problem we address in this paper is to find good networked storage strategies for answering *approximate data collection queries*. In particular, we want to ensure that a data collector who obtains access to (queries) *any* $(1 + \epsilon)k$ storage nodes is be able to recover at least $(1 - \delta)k$ original data packets. We will use linear codes over $GF(2)$ for distributed networked storage. Specifically, each data node will be routing its data packet to d randomly selected storage nodes who will be storing the bitwise XOR of what they receive. This has to happen before the data collection can take place and we will therefore refer to this routing of data packets to storage nodes as *pre-routing*. A data collector who queries $k(1 + \epsilon)$ storage nodes receives $k(1 + \epsilon)$ linear equations over $GF(2)$ and these can be hopefully used to recover $k(1-\delta)$ original data packets. We assume that the data collector has enough memory and processing power to store the equations and run the belief propagation algorithm (as will be explained in a subsequent section) to recover the original data.

Any linear networked storage strategy can be compactly represented as $s = mG$ where s is an $1 \times n$ vector of stored data, m is $1 \times k$ data vector and G is a $k \times n$ matrix with non-zero entries corresponding to the data packets that have been routed and combined in the storage nodes. Therefore, the desired properties of linear networked storage strategies can be translated into desired properties for the generator matrix G of a linear code.

Standard erasure codes can therefore be used for distributed networked storage in sensor networks. For example, Reed-Solomon codes or random linear codes [1] will produce storage strategies with $\delta = 0$ and $\epsilon = 0$.

However, the key issue is that both the data and the storage nodes are distributed, and the question we explore is how to build codes that are suitable for distributed network constructions. The fact that the code is created over a network introduces new requirements and constraints and therefore requires novel code design. In particular, every non-zero element in the generator matrix of the code, corresponds to a data packet that has to be pre-routed from a data node to a storage node. In fact, the communication required to construct the networked storage code is proportional to the *pre-routing degree* d of each data node, that is, the number of storage nodes that each data node has to pre-route its packet to. Therefore, one desired property is that the codes used for distributed networked storage have as sparse generator matrices as possible to minimize the required communication.

The second desired property is random and independent construction. Any requirement on the structure of the generator matrix would require data nodes to coordinate so that they route the correct packets to the appropriate storage nodes. We would ideally like to have each data node choosing which storage nodes to contact randomly and independently, or equivalently every row of the generator matrix created independently. This row independence, which we call "decentralized property", was proposed in our previous work [3, 4] and leads to stateless robust randomized algorithms for distributed networked storage.

1.3. Previous work and Contributions

The main question we address in this paper is related to the sparsity of G which is measured by *how d can scale as a function of the network size n* so that almost all the data can be recovered. In our previous work [3, 4] we demonstrated how to solve the (exact) distributed networked storage problem when the requirement was *to recover all k data packets by querying any k storage nodes*[1]. We introduced decentralized erasure codes which can be constructed by having each data node pre-route its data packet to $d = \Theta(\log n)$ randomly selected storage nodes and further showed how this logarithmic degree is optimal if each data node is acting randomly and independently. This however leaves open the question of what can be possibly achieved when the pre-routing degree is constant (does not grow with the network size n) if some form of coordination between the data nodes is allowed.

The main contributions of this paper are the following. We show that *any networked storage strategy* (even if data nodes are coordinated by some centralized authority) with constant degrees d will fail to recover all k data packets with at least a constant probability (section 2). Therefore, since recovering all the data packets with constant degrees is impossible, we relax the problem and investigate how to recover a linear fraction of the k data packets. We show that this problem can be solved with constant pre-routing degrees using a specific choice of fountain codes (section 3). The surprising fact that one can construct such sparse linear equations that can still be used to recover almost all the data stems from carefully designed degree distributions [11]. The last part of the paper (section 4) focuses on grid topologies and proposes a randomized algorithm to create the fountain codes over the network. A key step in the analysis of our algorithm is a novel result on the time until a random walk on a grid is absorbed, when there are randomly scattered traps.

[1] Which corresponds to setting $\delta = 0, \epsilon = 0$ in our problem setting

2. LOWER BOUND ON THE ERROR PROBABILITY FOR CONSTANT DEGREES

As mentioned, any deterministic or randomized linear distributed networked storage strategy can be described in terms of a linear code and its generator matrix G. In this section we show that networked storage strategies with constant pre-routing degrees that do not grow with n, will fail with at least a constant probability if we require that $\delta = 0, \epsilon = 0$. Assume that some data node pre-routes its packet to a *constant* number of storage nodes. This means that there exists a row in the generator matrix of the code that has only a constant number of nonzero elements. The probability that the networked storage code fails (when $\delta = 0, \epsilon = 0$) is the probability that k randomly selected storage nodes do not contain enough information to recover all k data packets. We present the following bound (Proof omitted due to space constraints):

Proposition 1. *For an (n, k) linear code where $n = qk$, $q > 1$, if there exists a row in the generator matrix that has no more than c constant nonzero elements, the probability that the corresponding networked storage fails is always larger than $(\frac{q-1}{q})^c$.*

3. FOUNTAIN CODES FOR APPROXIMATE DATA COLLECTION

Proposition 1 demonstrates that if we desire to have constant pre-routing degrees that do not scale with the network size, it is not possible reliably to recover all the k data packets by querying any k storage nodes. Therefore, one natural question to ask is: what is the best that can be achieved with constant pre-routing degrees. In this section we show how one can use a fountain code to recover a constant fraction of the k data packets with pre-routing degrees which are random but bounded by a constant almost surely. A fountain code [11] is created by a set of k input symbols and a degree distribution $\mathcal{D}$. Each encoded symbol of a fountain code is created independently as follows: First, a degree d is sampled from the distribution $\mathcal{D}$. Then d out of k input symbols are chosen uniformly and independently (without replacement) and the resulting output symbol is the bitwise XOR of the d selected input symbols. Fountain codes are decoded by running the belief propagation algorithm and the key technical challenge is the careful design of the degree distribution $\mathcal{D}$ so that this iterative decoding procedure succeeds. The first fountain codes where invented by Luby and called LT-Codes [10]. The degree distribution used is called the robust soliton distribution and has logarithmic degree. This would correspond to logarithmic pre-routing degree for our networked storage problem. Raptor codes [11] manage to reduce the degrees from logarithmic to constant by using an appropriate pre-code. This idea cannot be used for our problem however, since the standard pre-codes do not have sparse and independent generator matrices. However we can use the LT code used inside the Raptor codes and still recover a constant fraction of the original data. Specifically, if we use the degree distribution with

generating function:

$$\Omega_D(x) = \frac{1}{\mu+1}\Big(\mu x + \frac{x^2}{1\cdot 2} + \frac{x^3}{2\cdot 3} + \cdots \quad (1)$$

$$+ \frac{x^D}{(D-1)\cdot D} + \frac{x^{D+1}}{D}\Big). \quad (2)$$

where $D = \lceil 4(1+\epsilon)/\epsilon \rceil$ and $\mu = \epsilon/2 + (\epsilon/2)^2$ for any $\epsilon > 0$. then we can use the following result [11]:

Lemma 1. *There exists positive c (depending on ϵ) such that with an error probability of at most e^{-ck} any set of $(1+\epsilon/2)k+1$ output symbols of the fountain code with parameters (k, Ω_D) are sufficient to recover at least $(1-\delta)k$ input symbols via belief propagation decoding, where $\delta = (\epsilon/4)/(1+\epsilon)$.*

Therefore this fountain code can be used for approximate networked storage if one wants to query $(1+\epsilon/2)k+1$ storage nodes and recover $(1-(\epsilon/4)/(1+\epsilon))k$ data nodes. Each storage node can independently sample its degree d from the given distribution Ω_D and then *it needs to request d data packets randomly and independently*. Notice that since d is always smaller than the constant $D = \lceil 4(1+\epsilon)/\epsilon \rceil$ and therefore the total number of pre-routed messages per data node is bounded by a constant almost surely (since k/n is fixed). A data collector can collect any $(1+\epsilon/2)k+1$ encoded packets and run the belief propagation decoder which will require a number of iterations proportional to the average degree of Ω_D. Since this average is $\mathsf{E}d \approx \ln(1/\epsilon)$ [11], the decoding complexity will be only $O(\log(1/\epsilon)k)$ which is linear in k and therefore order optimal. Note that for this construction each storage node is picking its degree and which data nodes to contact. This property, which is sometimes called the rateless property of fountain codes, corresponds to having each column of the G matrix being created independently and is the transpose of the decentralized property [4] mentioned in the previous section. Therefore, to construct these fountain codes over a network we need a mechanism for storage nodes *to be able to find random data nodes* and request for their data packets which will be in turned routed back to them. Such a mechanism is presented in the next section for grid topologies.

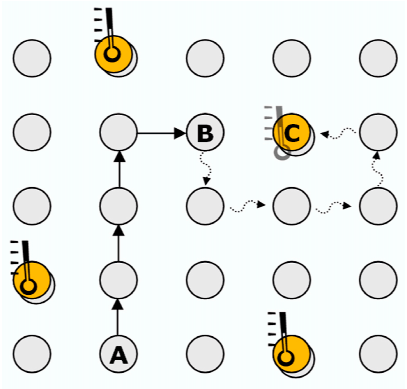

Fig 1. Example of the randomized algorithm for grids ($n = 25, k = 4$). The nodes with the thermometer are data nodes placed on the same grid site as a storage node. Here, storage node A wants to find a random data node. It selects a random location on the grid (the location of node B) and a request is routed greedily (solid arrows). Since there is no data node in site B, a random walk starts (dotted arrows) until data node C is found. In the example $R = 4, \mathsf{W} = 5$.

4. RANDOMIZED ALGORITHM FOR GRID TOPOLOGIES

In this section we address the problem of how to construct the fountain code over a grid topology with only local randomized decisions. We assume that our storage nodes are placed on the sites of a grid (of size $\sqrt{n} \times \sqrt{n}$) with wireless transmission radius large enough for the four nearest neighbor connectivity. Grid topologies of sensor networks have been analyzed and many results are known, see for example [2]. We further assume that the $k = qn$ data nodes are placed randomly on some sites of the grid (as an extra device placed on the same location with the storage node and operating independently). We also assume that each node knows its location on the grid.

We want to use the networked storage strategy proposed in the previous section but we need a mechanism to send requests to random data nodes who can then route back their data packet. After d data packets have been received the storage node can XOR them, store the result and essentially become a fountain code encoded symbol. To find a random data node, we propose the following scheme. First a randomly selected location of the grid is selected by the storage node who is going to initiate the request. Then, a request packet is routed to that random location using greedy geographic routing. Since every node knows their location this procedure can be performed with only local information and will terminate after R steps, and R is always $O(\sqrt{n})$. If that randomly selected site happens to have a data node, then that data node receives the request, routes back the data packet using greedy geographic routing and the procedure terminates. If however the randomly selected site is occupied only by a storage node, then the request initiates a (simple) random walk until it hits a site containing a data node. We assume that the $k = qn$ data nodes are randomly scattered, such that all the $\binom{n}{qn}$ ways of choosing their locations are equally probable. Therefore, *the data nodes are acting like traps for the random walk* and what we want to show is that the walk is trapped sufficiently fast. Specifically, let a random walk start from a uniformly selected position (which we assume not to be a data node). Let W denote the (random) number of steps before the random walk is trapped (i.e. hits a data node and terminates). Therefore, in the proposed protocol there are two phases: the routing phase (geographic routing to the uniformly preselected position) which takes $R = O(\sqrt{n})$ steps and the random walk phase which takes W steps. We want to show that

$$R + \mathsf{W} \simeq R \quad (3)$$

asymptotically almost surely (a.a.s.) for large n. This means that the cost of the random routing dominates and the average cost is nearly equal to the case where the measuring node *knew where to send the packet*.

Note that the actual probabilities that each data node receives the request under this algorithm *are random variables that depend on the realization of their locations*. For example if there is a cluster of data nodes somewhere, the nodes having only data node neighbors will be receiving the requests with lower probability relative to data nodes with many storage node neighbors. However, the expected reception probabilities are uniform and large fluctuations should not typically happen. In this paper we only analyze this expected behavior and in future work we plan to establish a concentration result around this expectation for large networks.

There has been significant work on problems related to random walks on infinite lattices with traps [6, 7], but to the best of our knowledge, there are no known results for the scaling behavior of the trapping time for finite lattices. Therefore the following result

might be of independent mathematical interest:

Theorem 1. *Let a random walk start from a uniformly selected position on a $\sqrt{n}$ by $\sqrt{n}$ grid. Exactly qn of the nodes are traps and all the $\begin{pmatrix} n \\ qn \end{pmatrix}$ possible trap configurations are equally likely. If W denotes the number of steps before the random walk hits a trap for the first time, then $\mathsf{W} = \mathsf{O}(\log n)$ a.a.s. in particular*

$$Pr(\mathsf{W} > \log n) = \mathsf{O}(\frac{1}{\log \log n}). \tag{4}$$

Proof: Due to space constraints, we present a sketch of the proof to highlight the techniques. We define an "inner region" $\mathcal{R}_{\text{inner}}$ inside the grid which has enough distance from the boundary so that a random walk that starts inside it will not be affected by the boundary before it takes $\log n$ steps.

$$\Pr(\mathsf{W} > \log n) \leq \Pr(\mathsf{W} > \log n | W_0 \in \mathcal{R}_{\text{inner}}) + 4\frac{\log n}{\sqrt{n}}$$

where the last inequality follows from the fact that all nodes have a probability of $1/n$ of being W_0 and there are less than $4\sqrt{n}\log n$ nodes in $\mathcal{R}_{\text{inner}}$. To bound the first term, let S_ℓ (called the *range of the random walk*) denote *the number of distinct sites visited by the random walk in ℓ steps*. If we know S_ℓ, the probability that the walk has not ended after $\ell \leq \log n$ steps is

$$\Pr(\mathsf{W} > \ell | W_0 \in \mathcal{R}_{\text{inner}}, S_\ell = s) = \frac{\begin{pmatrix} n-s \\ qn \end{pmatrix}}{\begin{pmatrix} n \\ qn \end{pmatrix}}. \tag{5}$$

Using Stirling's approximation, for $s \leq \log n$ and $\rho \stackrel{\text{def}}{=} s/n$, it is not hard to show that there exits a constant c_s such that $\Pr(\mathsf{W} > \ell | W_0 \in \mathcal{R}_{\text{inner}}) \leq c_s \sum_{i=1}^{\ell} (1-q)^i \Pr(S_\ell = i | W_0 \in \mathcal{R}_{\text{inner}})$.

The sum goes only up to ℓ because S_ℓ can never be larger than ℓ. Notice that random walks that start inside $\mathcal{R}_{\text{inner}}$ and take less than $\log n$ steps are indistinguishable from random walks on the infinite grid (since they are not affected by the boundary), so if $\tilde{S}_\ell$ is the range of the random walk on the infinite grid, we can write

$$\Pr(\mathsf{W} > \ell | W_0 \in \mathcal{R}_{\text{inner}}) \leq c_s \sum_{i=1}^{\ell} (1-q)^i \Pr(\tilde{S}_\ell = i) = c_s \mathsf{E}(1-q)^{\tilde{S}_\ell}. \tag{6}$$

The following results are known from Dvoretzky and Erdős [5], and Jain and Pruitt [8] for $\tilde{S}_\ell$.

$$\mu = \mathsf{E}[\tilde{S}_\ell] = c_1 \frac{\ell}{\log \ell} + \mathsf{O}\left(\frac{\ell}{(\log(\ell)^2)}\right). \tag{7}$$

$$\sigma^2 = \mathsf{Var}(\tilde{S}_\ell) \leq c_2 \frac{\ell^2}{(\log(\ell)^2)}. \tag{8}$$

Using Chebyschev's inequality for $\tilde{S}_\ell$ we get

$$\Pr(\tilde{S}_\ell \leq \mu - t\sigma) \leq \Pr(|\tilde{S}_\ell - \mu| \geq t\sigma) \leq \frac{1}{t^2} \tag{9}$$

Let $k(\ell) = \mu - t\sigma = c_1 \frac{\ell}{\log \ell} - c_2 t(\ell) \frac{\ell}{(\log(\ell))^2}$, where $t(\ell)$ is a free parameter of the Chebyschev bound. We choose $t(\ell) = \sqrt{\log(\ell)}$ So from (9), we obtain

$$\Pr(\tilde{S}_\ell \leq k(\ell)) \leq \frac{1}{t^2} = \frac{1}{(\sqrt{\log(\ell)})^2}. \tag{10}$$

Recall that the sum we want to bound is

$$\Pr(\mathsf{W} > \ell | W_0 \in \mathcal{R}_{\text{inner}}) \leq c_s \sum_{i=1}^{\ell} (1-q)^i \Pr(\tilde{S}_\ell = i). \tag{11}$$

This is the sum of the density of $\tilde{S}_\ell$ weighted with a decreasing function $(1-q)^i$. It is therefore clear that if *we shift some probability mass of the distribution of S_ℓ to smaller values of i, we will get a larger sum*. We define a new function $g(i)$ by shifting all the probability mass of $\tilde{S}_\ell$ that is smaller than $k(\ell)$ to 1: $g(1) = \frac{1}{\log(\ell)}$. The remaining mass $\Pr(\tilde{S}_\ell > k)$ is of course smaller than 1. So we may define $g(k(\ell)) = 1$ and let $g(i) = 0$ for all other values. We therefore have the key inequality:

$$\Pr(\mathsf{W} > \ell | W_0 \in \mathcal{R}_{\text{inner}}) \leq c_s \sum_{i=1}^{\ell} (1-q)^i \Pr(\tilde{S}_\ell = i) \tag{12}$$

$$\leq c_s \sum_{i=1}^{\ell} (1-q)^i g(i)$$

$$= (1-q)c_s \frac{1}{\log(\ell)} + c_s (1-q)^{c_1 \frac{\ell}{\log(\ell)} - \sqrt{\log(\ell)} c_2 \frac{\ell}{\log(\ell)^2}}.$$

If we choose the number of steps taken by the random walk as $\ell = \log n$ it is not hard to use the previous bound and obtain the desired result. ■

We have therefore shown that for grids each storage node can follow this simple randomized protocol to find random data nodes using $O(\sqrt{n})$ communication. Note that this is the diameter of the network and therefore the communication required to find a random data node is order optimal.

5. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewer for providing numerous suggestions for improving this paper and Prof. David Aldous of UC Berkeley for references and insightful discussions about the random walk problem.

6. REFERENCES

[1] S. Acedanski, S. Deb, M. Médard, R. Koetter , "How Good is Random Linear Coding Based Distributed Networked Storage?", NetCod 2005.

[2] G. Barrenetxea, B. Beferull-Lozano, and M. Vetterli. "Lattice Networks: Capacity Limits, Optimal Routing and Queueing Behavior". In IEEE/ACM Transactions on Networking, 2005.

[3] A.G. Dimakis, V. Prabhakaran, K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes" Proc. of IPSN 2005.

[4] A.G. Dimakis, V. Prabhakaran, K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage", Special Issue of IEEE Trans. on Inf. Theory: Networking and Information Theory. 2006.

[5] A. Dvoretzky and P. Erdős, "Some problems on random walk in space. Proc. of Second Berkeley Symp. Math. Statist. Probab. 1951, 353-367. Univ. California Press, Berkeley.

[6] W.Th.D. Hollander, "Random Walks on Random Lattices", Ph.D thesis 1985.

[7] Barry D. Hughes, "Random walks and random environments", Oxford University Press, (1996).

[8] N. C. Jain and W. E. Pruitt "The range of random walk." Proc. of Sixth Berkeley Symp. Math. Statist. Probab. 1972, 3 31-50. Univ. California Press, Berkeley.

[9] J. M. Kahn, R. H. Katz and K.S.J. Pister "Next century challenges: mobile networking for Smart Dust" Proc. of the 5th annual ACM/IEEE Int. Conf. on Mobile computing and networking, 271 - 278, 1999.

[10] M. Luby, LT Codes, In Proc. of IEEE FOCS, 2002.

[11] A. Shokrollahi, "Raptor Codes", Digital Fountain, Inc., Tech. Rep. DF2003-06-001, June, 2003