Network Coding for Distributed Storage in Wireless Networks

Alexandros G. Dimakis and Kannan Ramchandran

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

1 Introduction

We will address some of the problems related to storing information in multiple storage devices that are individually unreliable, and connected in a network. As an application consider a sensor network deployment in a remote and inaccessible environment where sensor nodes are taking measurements (possibly after processing) and storing data in the network, over long time periods. A data collector may appear at any location in the network and try to retrieve as much useful data as possible. Another scenario is a sensor network deployed in a time-critical or emergency situation (e.g. fire, flood, earthquake). Here, the focus is on maximizing the amount of sensed data than can be retrieved from a rapidly failing network. In both scenarios, many storage nodes are expected to fail and redundancy is necessary to guarantee the required reliability. This redundancy in the information representation can be introduced either through replication or through erasure coding. It is well known that information representations that use erasure codes require far less redundancy to provide the same level of reliability [42] and have been used in numerous applications (e.g. Reed-Solomon codes [37]). After extensive studies, essentially optimal erasure codes exist today, with linear encoding and decoding complexity [28, 40]. However, when coding is performed in an unstructured (and possibly dynamic) network, new issues arise that have not been addressed in classical coding theory. Specifically:

- *Communication* between storage and data nodes comes with a cost, since energy is a precious resource in sensor networks. Therefore, the code should be constructed with the minimal possible communication between nodes. This means that sparsity in the generator matrix of the code is critical for such applications.
- The information is sensed in multiple distributed locations and global *coordination* is difficult to achieve. Hence the code construction should be distributed and based on local knowledge.

- The sensor network will be deployed in a dynamic environment and the the encoded storage might need to *evolve over time*, to reflect such dynamic changes. For example when storage nodes are failing, new encoded packets need to be generated from existing encoded packets, naturally leading to network coding schemes.

In this chapter we discuss these issues and various related schemes that have been proposed in the recent literature including our own contributions. In summary, we will be interested in distributed, scalable and energy-efficient algorithms to generate and dynamically maintain encoded information representations in networks.

1.1 The Distributed Networked Storage Problem

We will be using the abstractions of a *data node* which is a *source* of information that must be stored, and a *storage node* which corresponds to a storage device with limited memory and communication capabilities. A physical sensor mote can have both sensing capabilities and sufficient memory, and hence can be both a data node and a storage node of our abstract model. This separation is useful because it simplifies the presentation and can be easily mapped back to actual devices.

The classical distributed storage problem consists of having multiple (distributed) storage nodes (e.g. hard disks) for storing data which is initially located at *one single data node* (see Figure 1).

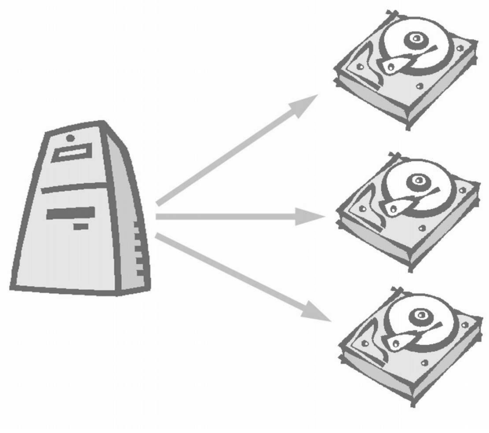

Fig. 1. The classical distributed storage setup. Data which is initially centralized is encoded and stored in distributed storage nodes.

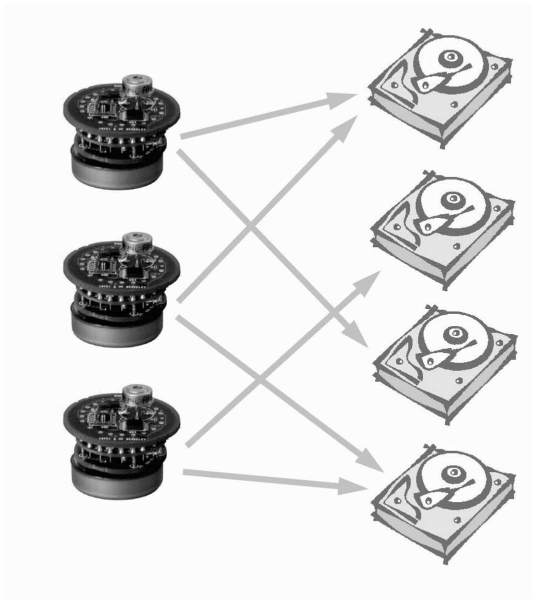

Fig. 2. Distributed networked storage. Initially distributed data stored in multiple storage nodes.

The *distributed networked storage* problem arises when both the data sources and the storage nodes are distributed (see Figure 2) and hence we have multiple data and storage nodes.

We make the following assumptions:

- We assume that there are k data-generating nodes and without loss of generality we will assume that each data node generates one data packet containing the information of interest.
- Further, assume we have $n \geq k$ storage nodes that will be used as storage and relay devices. Sensor nodes have limited memory, and we model that by assuming that each node can store only one data packet (or a combination having the same number of bits as a data packet). This is a key requirement to the scalability of the network. A data packet contains measurements over a time interval and can have significant size.
- The ratio $k/n = R$ (code rate) 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 generating data. These assumptions are only to simplify the presentation, and in practice the k data nodes and n storage nodes can be any arbitrary (possibly overlapping) subsets of nodes in a larger network.
- We are interested in schemes that require no routing tables, centralized processing or global knowledge or coordination of any sort. We rely on a packet routing layer that can route packets to *uniformly random locations*

in the network. Constructing such *random sampling algorithms* which are distributed and localized is key for the construction of codes in networks.

1.2 Outline

The chapter is structured as follows: in Section 2 we briefly present some properties of linear coding and network coding. Section 3 presents decentralized erasure codes and fountain codes and describes how their properties can be useful for storage in sensor networks. We also present randomized algorithms to construct such encodings with minimal coordination and knowledge. Section 4 discusses how encoded representations can be used for answering various queries other than data recovery. Finally, Section 5 discusses distributed network algorithms for randomly sampling sensor nodes with minimal communication; a mechanism that is necessary for constructing the codes described in this chapter.

2 Background

We will start with a brief survey of linear erasure codes and network coding. One important point is how the algebraic properties of the generator matrix of a code correspond to requirements from the network algorithm used to construct and maintain the encoded representation.

2.1 Linear Erasure Codes

Nearly all the coding schemes proposed in the literature are linear codes over finite fields. Erasure coding is a generalization of replication that divides the initial data object into k packets (or blocks) which are then used to generate n encoded packets of the same size. Good erasure codes have the property that *any* k out of the n encoded packets suffice to recover the original k data packets. In erasure coding we assume that the only types of errors that can happen are erasures of packets (due to failure of the corresponding storage node), but the packets that survive are always correct. Note also that we will be dealing with erasures of packets, not bits within a packet. Good erasure codes can yield much higher reliability compared to replication schemes for the same number of storage nodes. The most common erasure codes are Reed-Solomon codes, which are very widely employed in numerous applications like computer network distributed storage systems, and redundant disk arrays. Low-density parity-check (LDPC) codes and more recently Fountain codes [27] were proposed as alternatives with randomized construction and faster encoding and decoding times.

A toy example of a linear code over $GF(2^8)$ is given in Figure 3. In the example there are two data nodes X_1 and X_2 and three storage nodes Y_1, Y_2, Y_3.

We assume the data nodes have gathered a number of measurements. In the example we choose $u = 8$ bits to represent each number in our field which corresponds to $GF(2^8)$. The bits of the data measurements are divided into blocks of u bits which correspond to elements in $GF(2^8)$ (for example $X_1(1) = 002, X_1(2) = 080, X_1(3) = 220$). The data packet X_1 is routed to storage nodes Y_1, Y_3 and X_2 to Y_2, Y_3. Once a storage node receives one or more data packets, it must select coefficients f_i to multiply the received packets and subsequently add them to construct one encoded packet. A desired property is that the selection of the coefficients is done without any coordination, i.e. each storage node selects them uniformly and independently in $GF(2^8)$. Each coefficient then multiplies each block independently, multiple blocks are added (under the arithmetic of the Galois Field) and the results are cascaded into a new block packet Y_i that has exactly the same size as all the data packets. For example Y_3 has stored a packet that corresponds to $2X_1 + 1X_2$. Using this notation we mean that $Y_3(i) = 2X_1(i) + 1X_2(i)$ for $i = 1, 2, 3$. Each storage node will also store the coefficients f_i that it selected. This introduces an overhead storage that can be made arbitrarily small by coding over larger blocks [18, 11].

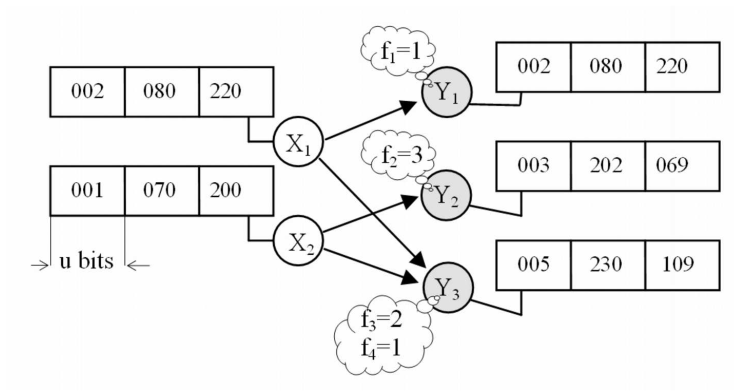

Fig. 3. A simple example of a linear code over $GF(2^8)$. Here $k = 2$, $n = 3, k/n = 2/3$, $q = 256$. The primitive polynomial of the field is $D^8 + D^4 + D^3 + D^2 + 1$. Arithmetic is done by representing numbers as binary coefficients of polynomials and doing polynomial operations modulo the primitive polynomial. For example, $70 \times 3 \rightarrow (D^6 + D^2 + D) \times (D + 1) = D^7 + D^6 + D^3 + D \rightarrow 202$.

Notice that in Figure 3 any two out of the three encoding packets can be used to reconstruct the original data.

In general, linear codes can be represented using their *generator matrix* in the form

$$s = mG, \tag{1}$$

where s is an $1 \times n$ encoded vector that is stored, m is $1 \times k$ data vector and G is a $k \times n$ matrix with elements selected from a field $GF(q)$. For the example in Figure 3,

$$G = \begin{pmatrix} 1 \; 0 \; 2 \\ 0 \; 3 \; 1 \end{pmatrix}. \tag{2}$$

To reconstruct m the receiver must invert a $k \times k$ sub-matrix G' of G. The key property required for successful decoding (that is also true for the example) is that any sub-matrix selection of G' forms a *full rank matrix*. When this is the case, decoding corresponds to solving a system of linear equations over $GF(q)$ (using for example, Gaussian elimination).

A matrix that has the property that *all* the square sub-matrices G' are full rank corresponds to an Maximum Distance Separable (MDS) code and such combinatorial constructions are quite difficult to achieve.

Reed-Solomon codes [37] construct such matrices by exploiting properties of polynomials over finite fields. The key idea is that any k interpolation points suffice to recover the coefficients of a degree $k-1$ polynomial. The smallest field size q for which MDS codes exist is unknown, and related to the MDS conjecture of algebraic coding theory:
(MDS Conjecture) Let G be a $k \times n$ matrix over $GF(q)$ such that every square sub-matrix G' is nonsingular. Then $q + 1 \geq n + k$.

A relaxation of this requirement (nearly-MDS) is that *almost all* the square sub-matrices G' are full rank, or equivalently that a randomly selected G' will be full rank with high probability. A *random linear code* over $GF(q)$ is the code generated by a matrix G that has each entry selected uniformly and independently from the finite field. It is well known [1] that the probability of a randomly selected G' being full rank can made arbitrarily close to one, by selecting a sufficiently large field size q.

2.2 Network Coding

Network coding is an exciting new paradigm for communication in networks where data packets are treated as entities which can be algebraically combined rather than simply routed and stored. The first major result [2] was a generalization of the max-flow min-cut theorem for multicasting. If there is one single source and multiple receivers, each receiver cannot hope to have throughput higher than the minimum cut separating it from the source, even if it was the only node being served. The theorem of Ahlswede et al. [2] states that if coding in the intermediate nodes of the network is allowed, *all the receivers* can have throughput equal to the minimum of the min-cuts separating each one from the source. In other words all the receivers can have the same throughput as the one with the weakest connection to the source, without limiting each other. It is easy to construct examples where such throughput cannot be achieved by simply routing packets from the source to the receivers.

Subsequently it was shown that linear coding suffices to achieve the multicast capacity [25, 24] and that random linear coding at intermediate nodes will suffice with high probability [18] for sufficiently large field size.

While most of the initial research on network coding focused on multicasting throughput, the fundamental idea of coding in intermediate nodes in networks has been shown to have advantages in other scenarios such as minimizing network resources [29], network diagnosis [44] and communication in wireless networks [23, 32, 45]. In this chapter we investigate applications of network coding for information storage, see [15] for a general introduction and other applications.

3 Coding for Networked Storage

3.1 Decentralized Erasure Codes

When trying to store linear combinations of data as information representations in sensor networks, new issues arise that make the existing codes unsuitable. Both random linear codes and Reed-Solomon codes have generator matrices that are *dense*, i.e. almost all the entries of G are non-zero. That means that every data node needs to send its packet to almost all n storage nodes to create the code generating $\Theta(n^2)$ communicating pairs (since $k = Rn$). A second desirable property is that the code can be created without coordination, and more specifically that each data node is choosing where to route its packet independently and also that the storage nodes are selecting their coefficients independently. Algebraically, this corresponds to having a code where every row of the generator matrix is created independently and is sparse. A code with this row independence property is called *decentralized* [11] and this property leads to stateless randomized network algorithms to generate the encoded information.

Randomized linear codes select every entry of G independently and therefore are decentralized [1]. However, they are not sparse and require significant communication to construct them. Algebraically the question is how sparse can a matrix with independent rows be made, and still have the property that square sub-matrices are full rank with high probability.

Decentralized erasure codes [10, 11] answer exactly this question: each data node routes its packet to $d(k) = c \ln k$ storage nodes. Each storage node selects random and independent coefficients f_i and stores a linear combination of the received packets. The main result of [10] is that $d(k) = c \ln k$ where $c > 5\frac{n}{k}$ is sufficient (and optimal up to constants) to ensure that randomly selected sub-matrices will be full rank with high probability. Decentralized erasure codes therefore have minimal data node degree and logarithmically many nonzero elements in every row.

Any erasure code can be decoded using Gaussian elimination in $O(k^3)$, but one can use the sparsity of the linear equations and have faster decoding.

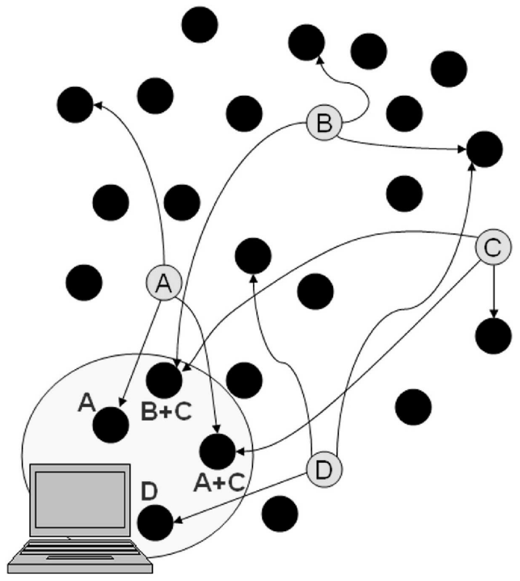

Fig. 4. Example of using linear codes for distributed storage. In this example there are $k = 4$ data nodes measuring information that is distributed and $n = 23$ storage nodes. We would like to diffuse the data to the storage nodes so that by accessing any 4 storage nodes it is possible to retrieve the data. Each data node is pre-routing to 3 randomly selected storage nodes. Each storage node has memory to store only one data packet so the ones who receive more than one packet store a linear combination of what they have received. The data collector in the example can recover the data by having access to (A, B+C, A+C, D).

Using the Wiedemann algorithm [43] one can decode decentralized erasure codes in $O(k^2 \log(k))$ time on average.

Randomized Network Algorithm

There is a very simple, robust randomized algorithm to construct a decentralized erasure code in a network: Each data node picks one out of the n storage nodes randomly and routes its packet to a randomly selected storage node. By repeating this process $d(k) = c \ln(k)$ times, we construct the decentralized erasure code. Note that we require a network layer mechanism that can route packets to randomly selected storage nodes in the network. Having a simple distributed mechanism that can perform this task with localized knowledge is key for many randomized algorithms and we discuss this issue in Section 5. Each storage node multiplies (over the finite field) whatever it happens to receive with coefficients selected uniformly and independently in $F(q)$ and stores the result and the coefficients. A schematic representation of this is given in Figure 5.

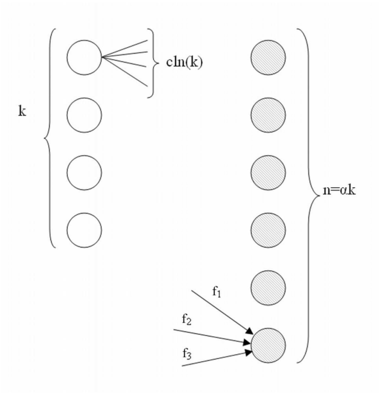

Fig. 5. Decentralized erasure codes construction. There are $d(k) = c\ln(k)$ edges starting from each data node and landing independently and uniformly on the storage nodes.

Storage Overhead

In addition to storing the linear combination of the received data packets, each storage node must also store the randomly selected coefficients f_i. The number of coefficients can be bounded by the number of balls that land into a bin when throwing $ck\ln(k)$ balls into n bins. It is standard problem in probabilistic analysis of algorithms [31] that the maximum load (the maximum number of coefficients a storage node will have to store) is $O(\log(k))$ with probability at least $1 - o(1)$. The total number of overhead bits to store the coefficients and data packet IDs is $O(\log(k)(\log(q) + \log(k)))$ which can be easily made negligible by picking larger data packet sizes. Notice that if we denote by $u = \log_2(q)$ the number of bits required to store each f_i, one can reduce the probability of error exponentially in the overhead bits.

Connections to network coding

An equivalent way of thinking of the distributed networked storage problem is that of a random bipartite graph connecting the k data nodes with the n storage nodes and then adding a data collector for every possible subset of size k of the n storage nodes. Then the problem of multicasting the k data packets to all the $\binom{n}{k}$ data collectors is equivalent to making sure that every collection of k storage nodes can reconstruct the original packets. This connection of storage and multicasting was proposed independently in [10, 21].

It has been shown that random linear network codes [25, 18] are sufficient for multicasting problems as long as the underlying network can support the required throughput. Decentralized erasure codes can therefore be seen as random linear network codes [18] on the (random) bipartite graph connecting the data and the storage nodes, where each edge corresponds to one routed packet. One key property is that in distributed storage, the communication graph does not correspond to any physical links but to virtual routing selections that are made by the randomized algorithm. Therefore this graph is not given, but can be *explicitly designed to minimize communication cost.* Essentially, we are trying to make this random bipartite graph as sparse as possible, while keeping the flow high enough and also allowing each data node to act independently. All the good codes described in previous sections have the property that they have very few edges ($o(n^2)$) connecting the data nodes and the storage nodes but can still guarantee very good connectivity between the any two subsets. Such bipartite graphs are called expanders [4] and are fundamental combinatorial objects for coding theory. It is easy to show that if one requires all $\binom{n}{k}$ data collectors to have k-connectivity with the data nodes, the corresponding bipartite graph needs to be dense. It is the probabilistic relaxation (a random data collector will have k-connectivity with high probability) that makes sparsity possible. This concept leads to *probabilistic expanders* that are formally defined and used for error correction in [8].

3.2 Fountain Codes

Fountain codes [27, 40] are linear codes over $GF(2)$ with sparse generator matrices and fast encoding and decoding algorithms. In particular, for LT codes [27], each encoded packet is created by first selecting a degree d from a carefully designed degree distribution (called the *robust soliton* [27]), and then taking the bitwise XOR of d randomly selected data packets. Therefore, fountain codes have the *rateless property*: every encoded packet is generated independently and there exists no predetermined rate since they can potentially generate an unbounded number of encoded packets. This corresponds to having every *column* of the generator matrix being independent and sparse (with logarithmic average degree similar to the decentralized codes). The degree distribution of the encoded packets is carefully designed so that a data collector who collects $k+\epsilon$ random packets (where the overhead ϵ is asymptotically vanishing for large k) can decode with a fast back-substitution algorithm which is special case of belief propagation [28]. Raptor codes [40] manage to reduce the degrees from logarithmic to constant by using an appropriate precode.

In this context, one can think of the decentralized property as being the "transpose" of the rateless property. This is because in decentralized codes, it is the rows of the generator matrix that are independent and this corresponds to having each data node acting independently. For sensor network applications, one implicit assumption is that it is easier for a data node to send its data to $d(k)$ randomly selected storage nodes than it is for a storage node to find and request packets from $d'(k)$ data nodes. This is true for many practical scenarios in which there are fewer data nodes that might also be duty-cycled or failing.

3.3 Partial data recovery

So far we have been addressing the problem of recovering all k data packets by querying k storage nodes. For this scenario, fountain codes are harder to create in networks, since creating the robust soliton degree distribution at storage nodes requires data node coordination. They however have the advantage of smaller field size (only binary operations) and lower computational complexity at the decoder ($O(k \log k)$ for LT codes versus $O(k^2 \log k)$ for decentralized codes). The pre-coding idea of Raptor codes cannot be easily performed over a network because it requires centralized processing.

Fountain codes can be used for partial recovery problems, where one is interested in querying fewer than k nodes and recover partial information. Creating a fountain code over a network where the data nodes are randomly located on a grid has been addressed in [12]. In this paper there is no pre-code, and the user is interested in recovering $(1-\delta)k$ data packets by querying $(1+\epsilon)k$ storage nodes. Random walks [26] can be used to create fountain encoded packets in sensor networks, to guarantee the persistence and reliability of cached data.

Sanghavi [38] investigated the optimal degree distribution for fountain codes when one is interested in recovering $(1-\delta)k$ data packets. Upper bounds on the performance of any degree distribution and lower bounds achieved by optimized distributions for any δ are presented in [38].

4 Information representations for query processing

The standard approach in query processing is to flood queries to all nodes, and to construct a spanning tree by having each node maintain a routing table of their parents. This is the approach currently used in both TinyDB and Cougar [30]. Flooding can be pruned by constructing an analog to indexes in the network, and an efficient indexing scheme is the Geographic Hash Table (GHT), which maps IDs and nodes to a metric space [36]. These approaches yield different tradeoffs between reliability over network changes, latency and communication cost. Coding can be used to add storage redundancy in any existing query processing scheme when high reliability or low latency is required.

A common type of query that can often appear in sensor network applications is an aggregate query, involving the average of a sensed quantity in a subset of the nodes. The first problem that arises is the in-network computation of such averages. The simplest algorithm is the construction of a tree-structure that averages the data of interest. Ganesan et al. [16] propose the DIMENSIONS system which uses wavelets to efficiently summarize and store sensor data in a natural hierarchical structure. When high fault-tolerance is required, or the complexity required to form a tree is too high, gossip and consensus algorithms constitute very simple distributed and robust alternatives [6, 13, 39]. Gao et al. [20] exploit the principle of fractionally cascaded information to provide efficient algorithms and theoretical bounds for answering range queries.

4.1 Exploiting Data Sparsity

For sensor network applications, the sensed data could be highly structured and this structure can be exploited to improve the performance. One approach for exploiting the structure of the sensed data can be used if we assume that the data is sparse in some basis that is known to the data collector. Recent results (see for example [14]) show that the actual storage devices can be ignorant of the sparse basis and simply make random projections that can be used to reconstruct by solving a linear program. Rabbat et al. [34] showed how gossip algorithms can be used to construct such random projections in a sensor network. Data collectors who obtain access to enough such projections by querying storage nodes can reconstruct the field by exploiting the underlying sparsity. Further, Wang et al. [41] showed how *sparse* random projections can be used and further guarantee a refinable approximation that improves as more sensors are queried.

4.2 Distributed Source Coding

If the statistical correlation structure of the data is known (or can be learned), distributed compression can be used to minimize the redundant information without having to collect the data in one location. Distributed Source Coding Using Syndromes (DISCUS) [33] is a practical means of achieving this. The data nodes form the syndromes of the data packets they observe under suitable linear codes. These syndromes are treated as the data which the nodes preroute to form the decentralized erasure codewords at the storage nodes. The data collector reconstructs the syndromes by gathering the packets from k storage nodes. Using DISCUS decoding, the collector can recover the original data from the syndromes. The correlation statistics, which is required by DISCUS can be learned by observing previous data at the collection point. The data nodes only need to know the rates at which they will compress their packets. This can be either communicated to them or learned adaptively in a distributed network protocol. The syndromes can be considerably shorter than the original data packets if the data observed by the different nodes are significantly correlated as is usually the case in sensor networks. Note that this approach is separating the source coding problem from the storage problem and this may not be optimal in general as shown in [35]. See also [46] for practical constructions.

4.3 Growth codes

In sensor network applications involving catastrophic or emergency scenarios such as floods, fires, earthquakes etc., the queries need to be adjusted to network dynamics. The setup is a rapidly failing sensor network where some nodes are sensing information that needs to reach the data collectors as soon as possible. Kamra et al. [22] show how fountain codes can be used for such applications and how the degree distribution needs to evolve over time to maximize the number of immediately recoverable data packets. Specifically, the authors design a dynamically varying degree distribution for partial network recovery to adapt to the data collector having received some data packets already and maximize the probability that the next packet is useful immediately. Growth codes initially create uncoded packets (since a data collector will have received nothing at the time and only degree one packets can be immediately useful). The degree distribution switches to pairwise XORs when the probability that a data collector already has a randomly selected packet becomes larger than the probability that the XOR cannot be decoded immediately.

4.4 The Repair problem

If the network is going to store data over long periods of time, with many nodes being duty cycled, it might be useful to monitor the storage nodes and actively refresh redundancy. When the number of storage nodes that actively respond falls below a threshold, fresh nodes (which might have been sleeping until that time) can be deployed to replace the failed ones and prolong data lifetime. However, the problem of creating new encoded packets in response to failures arises. When using replication, a new copy can be made from any other, but if the existing storage nodes are storing linear combinations of data, the problem of creating *new encoded packets from encoded packets* needs to be addressed. Regenerating codes [9] minimize the communication required to generate encoded packets from an existing encoded representation, to repair a failing network. In the same work, the minimal bandwidth required to repair any encoded storage scheme is computed explicitly using a flow formulation. Pyramid codes [19] are practical code constructions which trade space for partial recovery and efficient repair. While the repair problem has not been addressed in detail for sensor networks, simulations for distributed peer-to-peer storage systems suggest significant bandwidth savings over existing repair schemes.

5 Network Algorithms for Random Sampling

A very useful primitive operation in sensor networks is being able to find a (uniform) random node with small communication cost. All the coding constructions we have mentioned require such a mechanism, to route packets from data nodes to storage nodes. In addition, other applications like querying average sensor battery life, estimating the number of functional nodes and others could benefit from such a uniform sampling scheme [5].

If only local information is available at each node, it is not clear how to perform such a sampling without global knowledge. A simple idea is to accomplish this by performing a random walk in the network. We show that this is very costly for most relevant network topologies like grids and random geometric graphs. We further show that a simple greedy forwarding algorithm can approximately sample from random nodes using only local information and minimal communication.

5.1 Random Walks on Sensor Networks

Assume one wants to find a random node on a network by performing a random walk on the network nodes with properly adjusted transmission probabilities so that the invariant distribution is uniform over nodes. Clearly, after a few steps, one will be close to where the walk started and we need an estimate of how many hops are required before we have reached a truly random node. The number of steps required before the sampling distribution is reasonably close to uniform is measured by the *mixing time* of this Markov chain. To simplify the presentation assume we are dealing with a finite, irreducible and reversible Markov chain[1]. For two probability distributions θ_1, θ_2 defined on a finite space I, define the variation distance to be

$$\varDelta(\theta_1, \theta_2) = \frac{1}{2} \sum_{i \in I} |\theta_1(i) - \theta_2(i)|. \tag{3}$$

At time step t, assume that the Markov chain has some probability distribution of being at state j after t steps starting from state i: $P_{ij}(t)$. Since the MC is irreducible and aperiodic, it will have a limiting invariant distribution $\pi(j)$ and we know that $P_{ij}(t) \overset{t\to\infty}{\to} \pi(j)$. We are interested, assuming the worst case starting state i, to bound the distance of the distribution at time t to the invariant distribution. Define this distance to be

$$d(t) = \max_i \varDelta(P_{ij}(t), \pi(j)). \tag{4}$$

We define the mixing time (or "variation threshold time" [3]) to be

$$\tau = \min_t \{t \geq 0 : d(t) \leq \frac{1}{2e}\}. \tag{5}$$

It is therefore the first time where the Markov chain distribution becomes $1/2e$ close to the invariant, assuming the worst case starting state. The selection of

[1] Note that random walks on connected graphs are always in this class of Markov chains since self-loops can be added to cancel any periodic behavior.

the constant $1/2e$ is for algebraic convenience and any constant smaller than $1/2$ would work.

Therefore, if we use a random walk to sample from a sensor network, we need to perform $\Theta(\tau)$ steps before we are ϵ-close to the uniform distribution. The grid is the most simple model for a wireless sensor network topology, where n nodes are placed on a rectangular 4-connected square of edge length $\sqrt{n} \times \sqrt{n}$.

Unfortunately, even the 2-dimensional torus grid (which mixes faster than the regular grid) has a mixing time $\tau_{\text{Grid}} = \Theta(n)$ [3]. Therefore, one needs to perform $\Theta(n)$ random walk hops (visit approximately all the nodes in the network) to sample one random node.

Random Geometric Graphs

A random geometric graph $G(n, r)$ is formed as follows: place n nodes uniformly and independently in the unit square and connect nodes which are within distance r of each other (see Figure 5.1 for an example). Note that to simplify the analysis, some results rely on the assumption that the nodes are placed on the surface of a unit torus. Random geometric graphs have been established as standard models for wireless network topologies following the fundamental work of Gupta and Kumar [17] which shows that in order to have good connectivity and minimize interference, the transmission radius $r(n)$ has to scale like $\Theta(\sqrt{\frac{\log n}{n}})$.

Boyd et al. [7] investigate the question of the mixing time on $G(n, r)$ and establish that both the natural random walk (selecting each edge uniformly at random) and the fastest mixing reversible random walk (selecting the transition probabilities to minimize the mixing time) mix in:

$$\tau_{\text{RGG}} = \Theta(\frac{1}{r(n)^2}), \tag{6}$$

and for the critical radius $r(n) = \sqrt{\frac{\log(n)}{n}}$ we obtain

$$\tau_{\text{RGG}} = \Theta(\frac{n}{\log n}). \tag{7}$$

This slow mixing suggests that in random geometric graphs, it requires a very large number of hops to sample a random node.

5.2 Random Geographic Routing

We will now present a simple scheme that can sample an approximately uniform random node using only $O(1/r)$ hops which is equal to the diameter of the network and therefore order optimal. The key requirement is that we assume that all the nodes know their locations and the locations of their one-hop

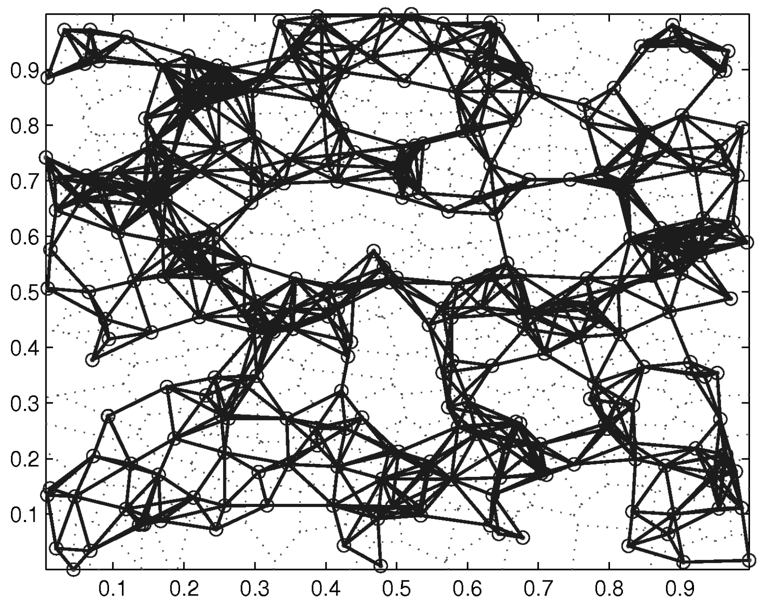

Fig. 6. Illustration of a Random Geometric Graph: solid lines represent graph connectivity, and dotted lines show the Voronoi regions associated with each node.

neighbors. Assume that some initial starting node s wants to find a random node in the network. The idea is to select a uniform geographic location in the unit square (called the target) and use greedy geographic routing towards that random target. Random geographic routing was proposed in [10] and independently in [5]. In [5] the authors further propose a rejection sampling scheme that can give sampling distributions that can be made very close to uniform. The performance of random geographic routing on random geometric graphs was analyzed in [13].

More formally, the random geographic routing scheme to sample a random node is the following:

1. Node s chooses a point uniformly in the unit square. Call this the target t. Node s forms the tuple $m_s = (l(s), t)$ where $l(s)$ is the geographic location of node s.
2. Node s sends m_s to its one-hop neighbor closest to t, if any exists. If node r receives a packet m_s, it sends m_s to its one-hop neighbor closest to t. Random geographic routing terminates when a node receives the packet and has no one-hop neighbor closer to the random target. Let v be that node, the output node sampled by the algorithm.

See [13] for a detailed analysis of the rejection sampling overhead for random geometric graphs and an application of random geographic routing for improving the convergence of gossip algorithms.

6 Conclusions

In this chapter we showed how erasure and network coding techniques can be used for storage in wireless sensor networks. The main conclusion is that sensor network applications introduce many novel challenges, mainly related to the distributed nature, as well as the communication and coordination constraints that naturally arise.

While some of these challenges have been addressed in the surveyed literature, numerous open problems remain. For example, the questions of combining the erasure encoding with multiresolution and distributed compression architectures, as well as faster encoding and decoding algorithms are among the issues that need to be addressed in future work. Distributed and scalable algorithms naturally fit with the randomized linear network coding theory and we believe that such ideas will be useful for practical applications.

References

1. S. Acedanski, S. Deb, M. Médard, and R. Koetter. How good is random linear coding based distributed networked storage. In *NetCod*, 2005.
2. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, 46:1204–1216, 2000.
3. D. Aldous and J. Fill. *Reversible Markov Chains and Random Walks on Graphs.* in preparation, 2005. (chapters available online), 2005.
4. N. Alon and J. Spencer. *The Probabilistic Method.* Wiley Interscience, New York, 2000.
5. B.A. Bash, J.W. Byers, and J. Considine. Approximately uniform random sampling in sensor networks. In *Proc. of the 1st Workshop on Data Management in Sensor Networks (DMSN '04)*, August 2004.
6. S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *Proceedings of the 24th Conference of the IEEE Communications Society (INFOCOM 2005)*, 2005.
7. S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Mixing times for random walks on geometric random graphs. In *Proceedings of SIAM ANALCO*, 2005.
8. C. Daskalakis, A.G. Dimakis, R. M. Karp, and M. J. Wainwright. Probabilistic analysis of linear programming decoding. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2007.
9. A.G. Dimakis, P.B. Godfrey, M.J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. In *Proceedings of IEEE Infocom*, 2007.
10. A.G. Dimakis, V. Prabhakaran, and K. Ramchandran. Ubiquitous Acess to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes. In *IEEE/ACM Int. Symposium on Information Processing in Sensor Networks (IPSN)*, April 2005.

11. A.G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. In *IEEE Transactions on Information Theory*, June 2006.
12. A.G. Dimakis, V. Prabhakaran, and K. Ramchandran. Distributed fountain codes for networked storage. In *Proceedings of IEEE ICASSP*, 2006.
13. A.G. Dimakis, A.D. Sarwate, and M.J. Wainwright. Geographic gossip: Efficient aggregation for sensor networks. In *IEEE/ACM Int. Symposium on Information Processing in Sensor Networks (IPSN)*, 2006.
14. D. Donoho. Compressed sensing. *IEEE Trans. Info Theory*, 52(4):1289–1306, April 2006.
15. C. Fragouli, J.Y. Le Boudec, and J. Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Comm. Review*, 2006.
16. D. Ganesan, D. Estrin, and J. Heidemann. Dimensions: Why do we need a new data handling architecture for sensor networks? In *Proceedings of ACM Computer Communication Rev., Volume 33, Number 1*, 2003.
17. P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.
18. T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, October 2006.
19. C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *IEEE International Symposium on Network Computing and Applications (NCA 2007)*, July 2007.
20. J.Gao, L.J. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *IEEE/ACM Int. Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.
21. A. Jiang. Network coding for joint storage and transmission with minimum cost. In *International Symposium on Information Theory (ISIT)*, July 2006.
22. A. Kamra, J. Feldman, V. Misra, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. *Proc. of ACM SIGCOMM*, 2006.
23. S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and Jon Crowcroft. XORs in the air: Practical wireless network coding. *Proc. of ACM SIGCOMM*, 2006.
24. R. Koetter and M. Médard. An algebraic approach to network coding. *Transactions on Networking*, October 2003.
25. S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Trans. on Information Theory*, 49:371–381, February 2003.
26. Y. Lin, B. Liang, and B. Li. Data persistence in large-scale sensor networks with decentralized fountain codes. In *Proceedings of IEEE Infocom*, 2007.
27. M. Luby. LT codes. *Proc. IEEE Foundations of Computer Science (FOCS)*, 2002.
28. M. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D. Spielman. Improved low-density parity check codes using irregular graphs. *IEEE Trans. Info. Theory*, 47:585–598, February 2001.
29. D.S. Lun, N. Ratnakar, M. Médard, R. Koetter, D.R. Karger, T. Ho, E. Ahmed, and F. Zhao. Minimum-cost multicast over coded packet networks. *IEEE Transactions on Information Theory*, June 2006.
30. S. Madden and J. Gehrke. Query processing in sensor networks. In *Pervasive Computing, 3(1)*, 2004.
31. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

32. D. Petrović, K. Ramchandran, and J. Rabaey. Overcoming untuned radios in wireless networks with network coding. *IEEE Transactions on Information Theory*, June 2006.
33. S. S. Pradhan and K. Ramchandran. Distributed source coding using syndromes (DISCUS): Design and construction. *IEEE Trans. Info. Theory*, 49(3):626–643, 2003.
34. M. Rabbat, J. Haupt, A. Singh, and R. Nowak. Decentralized compression and predistribution via randomized gossiping. In *IEEE/ACM Int. Symposium on Information Processing in Sensor Networks (IPSN)*, 2006.
35. A. Ramamoorthy, K. Jain, P.A. Chou, and M. Effros. Separating distributed source coding from network coding. In *42nd Allerton Conference on Communication, Control and Computing*, 2004.
36. S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data Centric Storage. In *Proceedings of the 1st ACM int. workshop on Wireless sensor networks and applications*, 2002.
37. I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. In *Journal of the SIAM*, 1960.
38. Sujay Sanghavi. Intermediate performance of rateless codes. *Information Theory and Applications (ITA)*, 2007.
39. O. Savas, M. Alanyali, and V. Saligrama. Randomized sequential algorithms for data aggregation in wireless sensor networks. In *Conference on Information Sciences and Systems (CISS)*, 2006.
40. A. Shokrollahi. Raptor codes. *IEEE Trans. on Information Theory*, June 2006.
41. W. Wang, M. Garofalakis, and K. Ramchandran. Distributed sparse random projections for refinable approximation. In *IEEE/ACM Int. Symposium on Information Processing in Sensor Networks (IPSN)*, 2007.
42. Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: a quantitiative comparison. In *Proc. IPTPS*, 2002.
43. D. H. Wiedemann. Solving sparse linear equations over finite fields. In *IEEE Transactions on Information Theory*, 1986.
44. C. Wu and B. Li. Echelon: Peer-to-peer network diagnosis with network coding. *Fourteenth IEEE International Workshop on Quality of Service (IWQoS)*, 2006.
45. Y. Wu. On constructive multi-source network coding. In *International Symposium on Information Theory (ISIT)*, July 2006.
46. Y. Wu, V. Stankovic, Z. Xiong, and S.-Y. Kung. on practical design for joint distributed source and network coding. In *Proc. 1st Workshop on Network Coding, Theory, and Applications (NetCod)*, April 2005.