Growth Codes: Maximizing Sensor Network Data Persistence*

Abhinav Kamra
Columbia University
New York, USA
kamra@cs.columbia.edu

Vishal Misra
Columbia University
New York, USA
misra@cs.columbia.edu

Jon Feldman
Google Inc.
New York, USA
jonfeld@google.com

Dan Rubenstein
Columbia University
New York, USA
danr@cs.columbia.edu

ABSTRACT

Sensor networks are especially useful in catastrophic or emergency scenarios such as floods, fires, terrorist attacks or earthquakes where human participation may be too dangerous. However, such disaster scenarios pose an interesting design challenge since the sensor nodes used to collect and communicate data may themselves fail suddenly and unpredictably, resulting in the loss of valuable data. Furthermore, because these networks are often expected to be deployed in response to a disaster, or because of sudden configuration changes due to failure, these networks are often expected to operate in a "zero-configuration" paradigm, where data collection and transmission must be initiated immediately, before the nodes have a chance to assess the current network topology. In this paper, we design and analyze techniques to increase "persistence" of sensed data, so that data is more likely to reach a data sink, even as network nodes fail. This is done by replicating data compactly at neighboring nodes using novel "Growth Codes" that increase in efficiency as data accumulates at the sink. We show that Growth Codes preserve more data in the presence of node failures than previously proposed erasure resilient techniques.

Categories and Subject Descriptors: E.4 [Coding and Information Theory]: Formal models of communication

General Terms: Algorithms, Design, Experimentation.

Keywords: Network Resilience, LDPC codes.

1. INTRODUCTION

One of the motivating uses of sensor net technology is the monitoring of emergency or disaster scenarios, such as floods, fires, earthquakes, and landslides. Sensing networks are ideal for such scenarios since conventional sensing methods that involve human participation within the sensing region are often too dangerous. These scenarios offer a challenging design environment because the nodes used to collect and transmit data can fail suddenly and unpredictably as they may melt, corrode or get smashed.

This sudden, unpredictable failure is especially troubling because of the potential loss of data collected by the sensor nodes. Often, the rate of data generated within a sensor network greatly exceeds the capacity available to deliver the data to the data sinks. With so many nodes trying to channel data to the sink node, there is congestion and delay in the neighborhood of the sink. This phenomenon can be described as a *funneling effect* [11], where much of the data trying to reach the sink is stalled. It is during this time that data is especially vulnerable to loss if the nodes storing the data are destroyed or disconnected from the rest of the network. Even if state-of-the-art compression techniques such as distributed source coding [33] or routing with re-coding are applied [8], there can be a significant delay between the time a unit of data is generated and the time at which it reaches the sink. We define the *persistence* of a sensor network to be the fraction of data generated within the network that eventually reaches the sink.

The goal of this paper is to investigate techniques to increase the persistence of sensor networks. To our knowledge, this is the first paper that concerns itself with this particular problem. Our solutions are based on the observation that even though there is limited bandwidth to forward data towards the sink, there still remains sufficient bandwidth for neighboring nodes to exchange and replicate one anothers' information. While such replication does not increase the rate at which data moves toward the sink, it does increase the likelihood that data will survive as some of the storage points fail.

We first focus on providing persistence in a sensor network that is deployed to take a "snapshot" reading of a particular region: each node's primary task is to take a single reading and relay this reading to a sink whose position (with respect to the node) is not necessarily known. This scenario is likely in disaster settings where getting an initial reading is essential, and nodes must be "dumped" into the region with limited or no planning and configuration, and where the topology may change rapidly due to node failures (burning up, getting crushed, etc.). Such networks can be thought of as "zero-configuration", where data collection and transmission must be initiated immediately. Hence, nodes have little opportunity to learn about specifics of the topology within which they are deployed, aside from some limited information describing their immediate surrounding area. Global information, such as the location

*This work was supported in part by NSF grants ANI-0238299, CNS-0435168, ANI-0133829, ITR-0325495 and research gifts from Microsoft, Intel and Cisco.

SIGCOMM'06, September 11–15, 2006, Pisa, Italy.

or direction of a sink, or the complete sensor network topology are unknown. Even if the nodes get to know the location and direction of sink(s) and are able to setup a routing tree, node failures will result in frequent disruptions and cause the routing setup to become invalid.

We design a novel data encoding and distribution technique that we refer to as a *Growth Code*. This code is designed to increase the amount of information that can be recovered at a sink node at any point in time, such that the information that can be retrieved from a failing network is increased. These codes are also easily implemented in a distributed fashion - another important criterion for sensor networks. The code grows with time: initially codewords are just the symbols themselves, but over time, the codewords become linear combinations of (randomly selected) growing groupings of data units. A well-designed code will grow at a rate such that the size of the codeword received by the sink is that which is most likely to be successfully decoded and deliver previously unrecovered data.

In a distributed sensor network where the nodes employ Growth Codes to encode and distribute data, the sink receives low complexity codewords in the beginning and codewords of higher and higher complexity later on. Identifying the optimal transition points at which the code switches to higher complexity codewords is a non-trivial task. We prove that such a code where the complexity of codewords increases monotonically is optimal in recovering the maximum amount of data at *any* time provided the transition points are carefully chosen.

We formally analyze Growth Codes and find close upper bounds for the transition points. In a *perfect source* setting, where the sink receives codewords that exactly fit a certain desired degree distribution, we compare the performance of a degree distribution based on Growth Codes with other well known distributions such as *Soliton* and *Robust Soliton* [18]. Furthermore, we simulate various distributed network scenarios (including some mimicking disaster scenarios) to evaluate the performance of Growth Codes. We find that in all the studied network scenarios, if the sensor nodes use the encoding protocol based on Growth Codes, the sink node is able to receive novel data at least as fast as any other protocol and in most cases, is able to recover *all* symbols in less than half the time compared to when no coding is used.

We then show how to extend growth codes to network settings in which nodes must make periodic measurements of the existing region. To this end, we provide the main features of a practical implementation of Growth Codes. Using these features, we also implement these codes on real mote-based sensor devices and perform experiments which show using Growth Codes results in much faster recovery of distributed sensor data.

The rest of the paper is organized as follows: The next section details some previous related work. In Section 3, we formulate the problem and describe the network setting in which it is employed. Section 4 describes the various solution approaches including the novel approach based on Growth Codes and the encoding/decoding algorithms used. In Section 5 we mathematically analyze the encoding protocol based on Growth Codes and prove interesting properties of the said protocol. We compare Growth Codes with other coding schemes in a perfect source setting in section 6. In Section 7, we describe the practical aspects of implementing Growth Codes. We present simulation and experimental results in Sections 8 and 9 to study the performance of our protocol in various sensor network settings. Section 10 describes how to handle sensor nodes which generate data periodically. We conclude in Section 11.

2. RELATED WORK

Much of the coding work in sensor networks targeted towards increasing the efficiency of data transmission to sink points utilizes *source coding*, which takes advantage of spatial and temporal correlations in data to compress the data to be delivered. Examples include systems like TSAR [25] and PRESTO [19] that use wavelet compression to reduce the overheads in transmitting correlated data. A local clustering scheme which is near-optimal across a range of spatial correlations is suggested in [32] although they ignore temporal correlations and losses in the network.

Our work, in contrast, can be viewed as a distributed *channel coding* where coding is performed to more effectively recover from stochastic losses of information (in our case, failing nodes). Most work in this area is focused on efficiently recovering *all* data, and for many of the schemes, data cannot be partially recovered. The canonical coding scheme is the optimal block erasure Reed-Solomon codes [17] which have been adapted for erasure-resilient data transfer [21]. The high computational costs for encoding and decoding these codes has pushed research in the direction of LDPC codes, such as the class of Digital Fountain codes [16], that include Tornado codes [20] and LT codes [18]. Byers et. al. optimize large transfers in [14] using codes by having the source re-code data as it learns from receivers what information they still require. Considine et. al. [15] propose a heuristic for generating good degree distributions which can be used to construct low complexity erasure codes. These schemes not only emphasize the recovery of all data, but are also centrally encoded, and cannot be trivially applied in settings where the data is distributed to begin with.

Distributed encoding schemes that aim to recover all data permit nodes to re-code data en-route to the destination have also been shown to be a more efficient means of communication [27]. A particular area of recent interest is the application of network coding within multicast environments (e.g., [29, 9], where it has been shown that simple linear codes are very efficient [24]. In [31, 5], a coding scheme is constructed to facilitate recovery of popular data by replicating coded versions of the data at multiple nodes. Katti et. al. [3] propose that nodes guess what data other nodes already have and exploit local coding opportunities to reduce the consumed bandwidth.

Some studies try to combine source and channel coding. In [23], a technique is presented that uses LDPC codes and source coding across two or three correlated sources to improve the energy efficiency of transmission in a sensor network. Marco et. al. [12] explore the tradeoff between efficient Slepian-Wolf coding and packet losses. They introduce variations in the Slepian-Wolf distributed coding to add some redundancy.

To our knowledge, the only work that focuses on partial recovery is *Priority Encoding Transmission* [4] which prioritizes data and provides reliable delivery of high priority data at low cost, at the expense of increased overhead to also recover low priority data. Drawbacks of this scheme in a sensor nets environment are the high recovery cost for much of the (lower priority) data, and that there is no known way to efficiently distribute the encoding process.

Directed Diffusion [10] uses dynamically chosen best paths and in-network data aggregation in distributed sensing environments to achieve energy and bandwidth efficiency.

3. PROBLEM SETUP

Our formulation consists of a sensor network whose general configuration is similar to that considered in a large body of work. We consider a network consisting of N sensor nodes where data sensed by a node describing properties of its immediate sensing region is

to be forwarded to a sink for further processing. The rate at which the nodes as an aggregate generate data may greatly exceed the communication capacity of the network such that forwarding becomes congested, and data must reside temporarily in the nodes before being delivered to the sink. Furthermore, the majority of the nodes cannot directly transmit to the sink, and hence other nodes must act as intermediate forwarding agents of data. These intermediate nodes have some computational power to manipulate data in transit, i.e., they can compress or recode data to increase delivery efficiency [34, 33].

There are also some specifics to our formulation that distinguish our environment from much previous work. We are concerned with monitoring dangerous emergency or disaster areas, such as earthquakes, floods, or fires. Often, nodes must be deployed quickly in response to the emergency, preventing an opportunity to statically optimize the deployment. Even if the network is planned beforehand, the disaster itself may significantly alter the network configuration, making the pre-planned configuration obsolete. These specifics translate to our problem as follow:

• Our initial study will assume that each node takes a single reading at a single point in time, and the network objective is to deliver as many of these readings as possible to the sink and as quickly as possible.

• The network is "zero-configuration", such that nodes must operate knowing only about their immediate sensing environment: the only topology information available to a node is its set of neighbors with whom it can communicate directly. This is either because the data is urgently needed, or nodes are expected to have extremely limited lifetimes. The latter property not only leaves limited time for the network to configure, but also induces rapid variation in topology, which would likely obviate any configuration that was constructed based on previous observations of topology.

• Our primary concern is the *persistence* of data: our goal is to minimize the amount of data lost due to failures of nodes as the emergency progresses (e.g., the fire spreads, rocks continue to bombard an area, or flood waters continue to grow). Traditional assumptions, like power conservation [22, 26] and optimized routing [7, 36] are secondary issues and are not addressed in detail in this paper.

3.1 Sensor Networks that Need Persistence

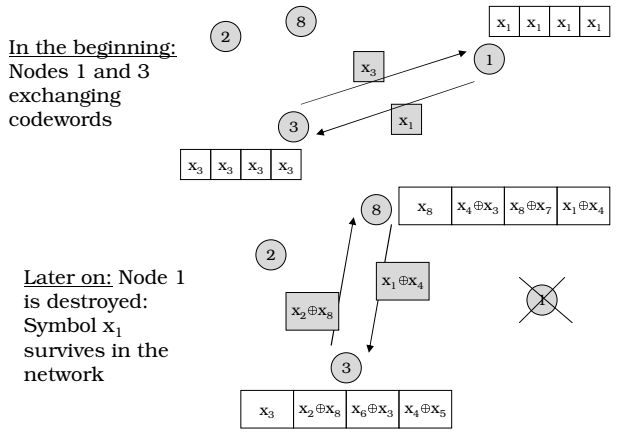

Figure 1: Localized view of the network. In the beginning, the nodes exchange degree 1 codewords, gradually increasing the degree over time. Even when a node fails, its data survives in the another node's storage

Figure 1 depicts the sensor network from the point of view of a single node at two different times. Each sensor node in the N-node network is arbitrarily placed to sense its data, and connects to a small subset of nodes with whom it can communicate directly. There are sink locations where data can be collected, but in a zero-configuration setting, these locations are almost always unknown to the sensing nodes. The arrows in the figure indicate the communication between nodes in an attempt to move the data, hopefully towards a sink. Initially the node has only copies of its own data, but over time, it will also contain (encoded) copies of other nodes' data, such that the failure of another node (in the figure, node 1) does not necessarily result in the loss of that nodes' data.

The type of disaster imposes a specific type of failure process of nodes in the network. For instance, in a fire or flood, one would probably expect node failures to be spatially and temporally correlated, where a node is more likely to fail if its neighbor has recently failed. In an earthquake or rock-storm, the failure process is probably best viewed as a sequence of randomly selected regions failing over time, where all nodes within a region are within close spatial proximity of one another and fail simultaneously, with different regions failing at independent times.

The failure of nodes in a region translates to a loss of memory in the global network, and any information stored in these failed regions is lost unless it is duplicated elsewhere or is already delivered to the sink. Data that is retrieved via the information collected at the sink is referred to as *recovered data*. In Figure 1, the nodes exchange *codewords* that consist of original data or XOR'd conglomerates of original data. Later on, even though a node fails, its data survives in the network.

At a high level, our goal is to try to find the best way to replicate information in the sensor network to maximize the quantity of recovered data at all times t, even as nodes fail in the network.

3.2 Network Description and Assumptions

Each node is equipped with some memory and processing power so that there is room for replication of information. The "best" way to replicate will depend on the capabilities of the various nodes, as well as various properties of the data (e.g., spatial and temporal correlation, priority, presumed accuracy). A sink node absorbs data at a rate λ_s, where λ_s depends on the number of sensor nodes to which the sink connects. In the network, the neighboring nodes can exchange data at a second rate λ_e.

Since we are making a preliminary foray into the problem of persistence, we believe it is most appropriate to thoroughly analyze the basic design of a persistence protocol in a somewhat constrained design space. Our analysis and description of the protocol makes the following assumptions about the sensor networking environment, though later in the paper, we extend the protocol and show experiments where these assumptions are relaxed.

• Node configurations are homogeneous: each node has a similar memory size C. The data collected at each sensor has an identical storage size of c, and we refer to this size-c data unit as a *symbol*. Hence, a node can ideally store up to $\lfloor C/c \rfloor$ symbols. We assume for now that the codewords also require size c[1].

• Data is not compressed, and all generated data is of equal importance. This assumption ignores the potential spatial correlations that often exist among sensed data [32, 13] as well as the likelihood that some data generated is more important than others (e.g., at nodes closer to regions of anomalous activity).

[1]In practice, some additional "header" space is necessary to describe the set of symbols encoded in the codeword. We address this issue in Section 7.

4. THE BASIC PERSISTENCE PROCEDURE

Having laid out our preliminary network model in the previous section, we are now ready to describe our basic persistence procedure. We divide the time into rounds. We emphasize that this division is merely to facilitate the description and evaluation of our techniques. The described techniques are easily extended to an asynchronous environment as we show in Section 7. In each round, neighboring nodes exchange information. Sink points attach to a subset of network nodes and are able to retrieve a fixed amount of information in each round.

- In every round, each node decides what information it must transmit to its neighbor, and how the incoming information from its neighboring nodes is stored.
- In every round, each sink collects new codeword symbols and *decodes* this information to retrieve the original data.

Initially, a node fills its memory with copies of its own data. In each round, a node has the opportunity to move or replicate codewords that exist in its memory to another node. Since the location of the sink is unknown, and since all data is of equal importance, we use a simple algorithm where a node randomly chooses a neighbor and a random codeword in its memory and exchanges this codeword with one of the codewords stored in the selected neighbor's memory. The only caveat to this exchange is that each node keeps a copy of its original data (this copy will be used in the encoding process described later). Periodically (perhaps once every M rounds, or perhaps M times a round), the sink samples codewords from nearby nodes. Unless otherwise specified, we assume $M = 1$, i.e., that $\lambda_s = \lambda_e$.

To increase the efficiency of data recovery at the sink, nodes can choose to *encode* the data to be sent. We consider several coding variants:

No coding: Nodes simply exchange the original symbols. The sink is likely to receive many duplicates. In the well studied *Coupon Collectors' Problem* [30], it has been shown that if the N original symbols are generated uniformly randomly, the sink needs to receive approximately $O(NlogN)$ symbols to recover all N original symbols.

Random Linear codes [35]: Here, the N original symbols are assumed to be elements of some finite field and each codeword is a linear combination of all N symbols, with the coefficients also being members of the same finite field. However, the encoding/decoding computational complexity of these codes is very high and hence, as mentioned in Section 2, they are impractical in our sensor network setting.

Erasure codes: Nodes can store and exchange codeword symbols formed using erasure codes. These can either be optimal erasure codes such as Reed-Solomon [17] or erasure codes based on sparse bipartite graphs such as Tornado [20] or LT codes [18] which trade efficiency of the code for fast encoding/decoding time. Considering the low computational power that sensor nodes have and the limited time they will have available to perform the encoding, only codes like LT Codes which have fast encoding/decoding algorithms are practical in our scenario.

Each codeword is an XOR of a subset of the N original symbols. The number of symbols which are XOR'd together is referred to as the *degree* of the codeword. These codes choose the degree of each codeword to fit a pre-determined *degree distribution* π which is a probability distribution over the degrees lengths.

A simple XOR encoder based on the degree distribution π works as follows:

1. Choose a degree d, where $d = i$ with probability π_i.
2. Randomly choose d distinct symbols out of the N symbols $x_1, \ldots, x_N$. Construct the codeword symbol by XOR-ing the d symbols.

It has been shown that if the sink receives an expected number of little more than N such encoded symbols, it can decode all of the N original symbols with high probability. One of the simple degree distributions used in LT codes is the *Soliton* distribution [18] which requires the sink to receive exactly N codewords to recover all N symbols. Unfortunately, this distribution has a high variance which means that there is a non-negligible probability that the sink will need many more codewords in practice. *Robust Soliton* is a modified form of Soliton distribution and performs much better in practice. We emphasize two drawbacks in applying these codes to our domain here:

• It is not straightforward to implement the encoding process in a distributed setting such that the appropriate degree distribution is obtained.

• These codes are not designed to maximize the amount of information recovered by the sink when only a small number of codewords are received, but are attempting to minimize the number of codewords needed to retrieve *all* data. We will show that Growth Codes out-perform their Soliton counterparts in this aspect.

4.1 Growth Codes

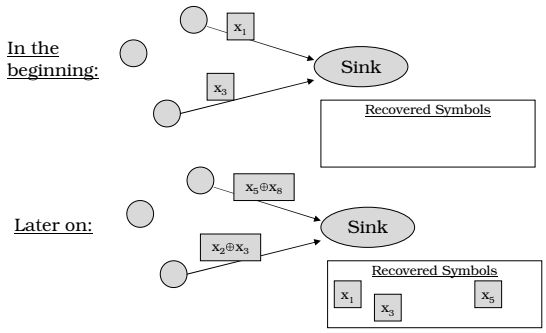

Figure 2: Growth Codes in action: The sink receives low degree codewords in the beginning and higher and higher degree later on

Growth Codes is a linear coding technique that encodes information via a completely distributed process. The technique not only ensures that the sink is able to recover *all* N of the data from only a little more than N codeword symbols, but also that data is efficiently recovered when only a small number of codewords is received.

We motivate the development of Growth Codes using the following toy problem. Consider a sink that is attempting to collect data over a series of rounds. The data can be XOR'd together to generate fixed-size codewords. Suppose the sink can select the degree[2] of the arriving codeword, but the set of symbols that form the codeword are selected uniformly at random. Also suppose the sink's goal at any time is to maximize the expected number of symbols it has recovered by that time. At any time t, given the collection of codewords the sink has already received, what should the sink choose as the degree of the next arriving codeword?

Clearly, for the first codeword, the degree should be one. Such a codeword will produce one symbol whereas any codeword of

[2]Once again, the degree of a codeword is the number of symbols XOR'd together to form the codeword

higher degree will produce no symbols. If the total number of symbols N is large, then the sink will also do well to request that its second received codeword be of degree one. If the sink continues to request codewords of degree one, eventually a point will be reached where a codeword of higher degree will very likely be decodable given the information received, and another codeword of degree one will very likely contain data that has already been received. It is not hard to follow this logic through to see that as more codewords are received, the "best" degree for the next codeword continues to grow. We will explore this phenomenon more formally in the next section.

Figure 2 depicts the working of the Growth Codes protocol. The sink node receives low degree symbols in the beginning from which it is able to immediately recover data. Later on, the sink received higher degree symbols which are more efficient in recovering more data.

In a completely distributed sensor network setting such as ours, where the nodes have limited storage and meager computational resources, a suitable encoding protocol is one which is easily implementable by the nodes and which is also efficient in terms of the number of symbols that can be recovered by the sink for a given number of received codewords.

Growth Codes are novel in that the codewords used to encode a fixed set of data change over time. Codewords in the network start with degree one and "grow" over time as they travel through the network en-route to the sink. This results is the the sink receiving codewords whose degree grows with time. Using such a design, if the network is damaged at any time and the sink is not able to receive any further information, it can still recover a substantial number of original symbols from the received codewords.

In order to facilitate formal specification of the procedure for encoding and decoding Growth Codes, we must first introduce some terminology we will use:

DEFINITION 4.1. *Given a set X of original symbols and a codeword symbol s of degree d, the* distance *of s from set X, $dist(s, X)$, is the number of symbols out of the d symbols XOR'd together to form s, which are not present in X.*

The decoder, *D*, that we use for our Growth Codes (to be implemented at the sink) will perform its decoding in an identical fashion to what is traditionally used in all Sparse Bipartite graph-based coding schemes such as LT Codes and Tornado Codes. This will allow us to fairly compare the performance of Growth Codes to prior coding schemes:

- Let A be the set of codewords received and X be the set of symbols already decoded (where X is initially 0).
- If there exists any codeword s for which $dist(s, X) = 1$, then decode the missing symbol encoded in s (by XOR'ing s with the symbols in X that were used to generate s), and add that symbol to X.

The decoding process used by the sink is an online version of decoder *D* such that the codewords are decoded on-the-fly as they are received and the codewords which cannot be decoded at the moment are stored for later use. We note that symbols whose distance is greater than 1 from X could potentially be combined to extract data symbols using sophisticated but more computationally expensive techniques (e.g., gaussian elimination).

A sequence of increasing values, $K_1, \ldots, K_N$ are hard-coded into the nodes prior to their deployment. The value of K_i indicates the point in time (i.e., the number of rounds after the data was generated) where such data should be encoded in codewords of degree no less than i. If after time K_i, a codeword of degree smaller than i is to be exchanged with a neighbor, the sending node will XOR the codeword with its own data symbol prior to exchanging it. In case the codeword already contains the data symbol of the sending node, nothing is done and the codeword is exchanged as is. Eventually, after being exchanged from node to node, the codeword will be exchanged by a node whose data symbol is not contained in the codeword. Then, that data symbol can be XOR'd into the codeword and its degree increased.

5. GROWTH CODES FOR DISTRIBUTED ENCODING AND ONLINE DECODING

In this section we prove some basic properties of Growth Codes which are critical in finding good values for the degree transition points, $\{K_i\}$, the points in time where the code should increase in degree. The "best" time for a network depends not only on the way the coders are implemented, but also on the topology of the network. Since we are designing codes for a zero-configuration network, topology information is not available. Hence, our codes will be designed under the assumption that when the sink receives a codeword of degree d, the data contained in that codeword is uniformly drawn at random. If the sensor network performs a good "mixing" of data prior to the data reaching the sink, this assumption is not unreasonable.

Solving for the optimal transition points is non-trivial for the decoder *D* described in the previous section. The difficulty is that the decoder maintains a memory of previously received codewords that could not be used at the time of their arrival, but can be used later on as further codewords arrive and additional data symbols are decoded. For tractability of the analysis, we will consider a restricted decoder that throws away any codewords it receives which cannot be decoded immediately, and does not maintain them for later use.

Due to lack of space, all the proofs are omitted but are available in our technical report [6].

5.1 Restricted Decoder

We define a decoding algorithm that is less powerful than decoder D. One might never use this decoder in practice, but it is used here since it is simpler to analyze and will help us prove bounds on the transition times K_i of Growth Codes.

A decoder is fed a sequence of symbols, $s_1, s_2, \ldots, s_k$. For a decoder α, we define $\alpha(s_1, s_2, \ldots, s_k)$ to be the number of symbols that α can decode when fed the sequence $s_1, s_2, \ldots, s_k$ in that order.

DEFINITION 5.1. *Decoder* S*, given a fixed sequence of codeword symbols $s_1, s_2, \ldots, s_k$, works as follows: Initially the set X of recovered symbols is empty.*

1. *Let $i = 0$.*
2. *From the remaining symbols, choose symbol s_i of the lowest degree. If $dist(s_i, X) = 1$, decode a new symbol and add to set X.*
3. *If $dist(s_i, X) = 0$ OR $dist(s_i, X) > 1$, throw s_i as unusable.*
4. *Increment i and go to step 2. Repeat until all symbols have been considered.*

Decoder *S* is more constrained than decoder *D*: *S* considers codewords in a fixed order only and throws away all codewords which cannot be decoded immediately (but could be of use to *D* at a later point in the decoding process). More formally, this can be stated as follows:

LEMMA 5.2. *Given a sequence of symbols $\sigma = s_1, s_2, \ldots, s_k$, and a reordering of these same symbols $\sigma' = s'_1, s'_2, \ldots, s'_k$. Then $D(s_1, s_2, \ldots, s_k) \geq S(s'_1, s'_2, \ldots, s'_k)$ for any sequence σ and any reordering σ' of that sequence.*

Decoder *S* works on a sequence of symbols sorted in non-decreasing order of their degrees. One can also view *S* as a decoder operating in an environment where codewords arrive in the order of non-decreasing degree.

5.2 Finding the best-degree codewords for decoder *S*

Let us consider our decoder *S* which does not maintain previously received codewords. When attempting to decode, it can only utilize the data that it has received previously and the next arriving codeword. The following lemmas address the "best" degree that the next codeword should have as a function of the number of already-decoded data symbols.

LEMMA 5.3. *Let $\rho_{r,d}$ be the probability of successfully decoding randomly chosen a degree d symbol when r symbols have already been recovered. Then $\rho_{r,d} = \frac{\binom{r}{d-1}\binom{(N-r)}{}}{\binom{N}{d}}$.*

Let R_i represent the number of symbols recovered by a sink when codewords of size greater than i provide a greater likelihood for providing recovery than those of degree less than i. We will show that, for decoder *S*,

$$R_1 = \frac{N-1}{2}, \ldots, R_i = \frac{iN-1}{i+1} \forall_{i \in [1, N-1]}. \quad (1)$$

LEMMA 5.4. *$\rho_{r,i} \geq \rho_{r,i+1}$ as long as $r \leq R_i = \frac{iN-1}{i+1}$*

The above result validates our earlier intuition that when the number of recovered symbols is small, low degree codewords are better and as the number of recovered symbols increases, higher degree codewords are better.

The result proves that before $R_1 = \frac{N-1}{2}$ symbols have been recovered, degree 1 codewords are the most useful. After that until R_2 symbols have been recovered, degree 2 codewords are the most useful and so it goes.

LEMMA 5.5. *If $\rho_{r,i} < \rho_{r,j}$ for some $i < j$, then $\rho_{r',i} < \rho_{r',j}$ for any $r' > r$*

This result basically states that once degree j codewords have become more useful than lower degree codewords, they will remain so even when more symbols are recovered. This monotonicity property is essential for Growth Codes which start with low degree codewords and switch to higher and higher degree codewords with time.

5.3 Properties of Decoder *D*

Now we consider some properties of the offline version of decoder *D*, which can then be applied to the online version of decoder *D* to help design Growth Codes.

THEOREM 5.6. *When using decoder* D *to recover r symbols, such that $r \leq R_1 = \frac{N-1}{2}$, the optimal degree distribution has symbols of only degree* 1.

THEOREM 5.7. *When using decoder* D *to recover $R_1 = \frac{N-1}{2}$ data units, the expected number of encoded symbols required is $K_1 = \sum_{i=0}^{R_1-1} \frac{N}{N-i}$.*

Theorems 5.6 and 5.7 show that if most of the network nodes fail and a small amount of the data survives, then not using any coding is the best way to recover maximum number of data units. We generalize these proofs in Theorems 5.8 and 5.9.

THEOREM 5.8. *To recover r symbols such that $r \leq R_j = \frac{jN-1}{j+1}$, the optimal degree distribution has symbols of degree no larger than j*

Let

$$K_j = K_{j-1} + \sum_{i=R_{(j-1)}}^{R_j - 1} \frac{\binom{N}{j}}{\binom{i}{j-1}(N-i)} \quad (2)$$

THEOREM 5.9. *To recover $R_j = \frac{jN-1}{j+1}$ symbols, at most K_j symbols are required in expectation*

5.4 A New Degree Distribution based on Growth Codes

According to the analysis in section 5.3, we observe that it is best to use only degree 1 symbols to recover the first R_1 symbols, only degree 2 symbols to recover the next $R_2 - R_1$ symbols and so on. Furthermore, an expected number of K_1 encoded symbols are required to recover R_1 symbols, an expected maximum K_2 codewords (recall that K_1 is the exact expected number while $\{K_i, i > 1\}$ are upper bounds on the expected number of codewords required) are required to recover R_2 symbols and so on.

This suggests a natural probability distribution on the degrees of the encoded symbols. In particular, if we need to construct a total of k encoded symbols, we should have K_1 degree 1 symbols so that we can recover an expected R_1 symbols from them, $K_2 - K_1$ degree 2 symbols so that we can recover an expected $R_2 - R_1$ symbols from them and so on as long as the k symbols are not yet received. A degree distribution can thus be defined as

$$\bar{\pi}^*(k) : \pi_i^* = max(0, min(\frac{K_i - K_{i-1}}{k}, \frac{k - K_{i-1}}{k})) \quad (3)$$

We call this the *Growth Codes degree distribution*.

In the encoding and data exchange protocol based on Growth Codes, sensor nodes can construct codewords that fit the desired degree distribution $\bar{\pi}^*(k)$. By choosing the degree transition points as $K_1, K_2, \ldots$ etc, the nodes generate codewords according to the distribution $\bar{\pi}^*(k)$ where k varies with time. If a sink node receives 1 codeword per round, it will receive degree 1 codewords for the first K_1 rounds, followed by degree 2 codewords until round K_2 and so on. If there are multiple sink nodes and they receive many codewords per round, then just by scaling the values of K_i, the desired effect can still be easily achieved.

6. PERFECT SOURCE SIMULATION MODEL

Growth Codes degree distribution as defined in Equation 3 grows monotonically in complexity with time and hence gives itself nicely to an implementation in a distributed environment (where the source data is distributed to begin with) as described in Section 4.1. It is difficult to compare Growth Codes with other well known degree distributions such as Soliton since it is not clear how these distributions can be implemented in a distributed scenario. The data symbols are inherently distributed to begin with and it is impossible to generate a truly random codeword of a specific degree unless all symbols are first aggregated at a single node. This problem is

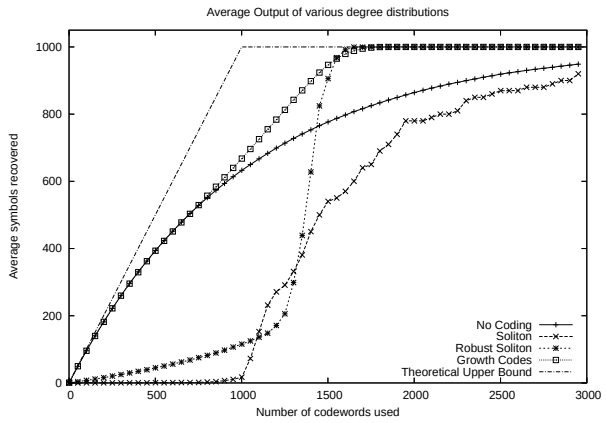

Figure 3: Comparing the performance of various degree distributions in a *perfect source* setting. N is chosen to be 1000

bypassed in the Growth Codes protocol by gradually increasing the degrees of codewords generated in a completely distributed manner.

For sake of comparison, let us assume it is possible to implement any degree distribution in a distributed setting so that the sink receives codewords exactly following this distribution. We now evaluate how the Growth Codes degree distribution compares to other well know degree distributions in such a scenario. We call this the *perfect source* simulation model since we generate codewords according to a specific degree distribution and assume that all these codewords are received at the sink.

We have the following scenario: There is one source and one sink. The source has all symbols $x_1, \ldots, x_N$. It constructs codewords according to some degree distribution π, i.e., a codeword of degree d is constructed with probability π_d and every codeword of degree d with uniform probability. The sink receives the codewords one by one and decodes them on the fly. If the sink cannot decode a codeword at any time, it is stored for later use when more symbols have been recovered. Note that the Growth Codes degree distribution as defined in Equation 3 changes with time, so that higher and higher degree codewords are generated as time goes on. We want to evaluate how much data can be recovered by the sink for a given number of received codewords. The sink uses the online version of decoder *D* as described in Section 4.1 to recover data on the fly.

We compare with the Soliton and Robust Soliton distributions where the values of Robust Soliton parameters c and δ are chosen to be 0.9 and 0.1 respectively as suggested by the authors in [18]. To assess the advantage of using coding if at all, we compare with a no coding scheme where each codeword is a randomly selected degree 1 symbol.

Figure 3 depicts the number of symbols recovered at the sink for any given number of codewords received according to the corresponding degree distribution. The results are from a mean of 100 simulation runs. The topmost curve plots the theoretical bound on the maximum number of symbols that could be recovered by that point in time (i.e., the $\min(k, N)$) where k is the number of codewords received at any point.

If no coding is used, the sink is able to decode a substantial number of symbols in the beginning. As more symbols are recovered, the Coupon Collector's effect kicks in and the no coding is not as good any more. Soliton and Robust Soliton accumulate a lot of high degree codewords which cannot be decoded until a substantial number of symbols have been recovered. This is why we see a sudden jump in the curve for Robust Soliton. For maximum data recovery at any time, it is easy to see that Growth Codes perform the best.

7. PRACTICAL IMPLEMENTATION OF GROWTH CODES

Up to now, we have made some rather strong assumptions about the sensing network environment to facilitate the presentation and evaluation of Growth Codes. Here, we address several of the limitations

7.1 Removing the notion of a "round"

Our previous description of Growth Codes required nodes to exchange information over a series of rounds. There is, however, no explicit need for synchronizing multiple exchanges. All that is necessary is that the degree distribution "grows" at approximately the right rate. The rate at which the codes grow (especially initially) is slow enough that even if the time at which nodes initialize (i.e., sense their data and attempt to exchange data with other nodes) varies, we expect little variation in the results. Our TOSSIM-based simulations and experiments demonstrate this claim.

7.2 Coefficient Overhead

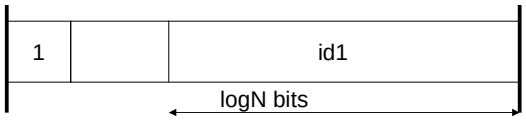

Figure 4: Structure of a coefficient when no coding is used

As alluded to earlier, each codeword, regardless of whether it contains original (degree one) symbols or a bona fide XOR of original symbols, must include a coefficient that descibes the symbols from which it is formed. We assume that each sensor node has a unique ID, and that it prepends this ID to its symbol. A codeword of degree i must therefore contain i of these IDs for the decoder (sink) to know how to decode the packet.

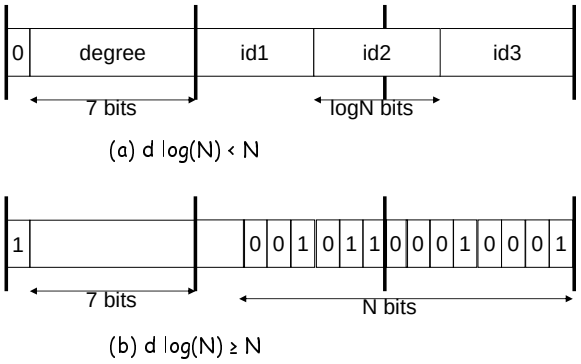

Figure 5: Structure of a coefficient with multiple degree codewords

- **No coding:** When no coding is used, each codeword is always degree 1. As shown in Figure 4, only a single ID is necessary to specify the coefficient. The first bit is always 1 and the last $log(N)$ bits specify which symbol makes up the degree 1 codeword.

- **With Coding:** When codewords can be more than degree 1, they can be specified in two ways: in a bit format and in $log(N)$ format. The first byte specifies how the remaining bytes are constructed. The first bit of the first byte is 1 if the coefficient is specified in $log(N)$ format and 0 when it is specified in the bit format.

  When the degree d of a codeword is low (in particular, less than $\frac{N}{log(N)}$), less space is consumed by listing the IDs of the d symbols that make up a codeword. In this case, the coefficient is constructed in the format as shown in Figure 5(a). The d symbol identifiers each of size $log(N)$ are packed into as few bytes as possible.

  When the degree is high (higher than $\frac{N}{log(N)}$), less space is consumed by reserving a bit for each of the N possible symbols that form a codeword. As shown in Figure 5(b), a 1 bit signifies the presence of a particular component and 0 specifies the absence.

If the size of a symbol is large, then these overheads are negligible. However, they are clearly less negligible when the data sizes are small. However, we note that Growth Codes grow slowly, and the degree of the codeword is, for the majority of the time, fairly small. In Section 10, when we consider how to forward from nodes that generate data periodically, we will demonstrate how to further reduce the amortized cost of these coefficients.

8. EXPERIMENTS ON A GRID NETWORK

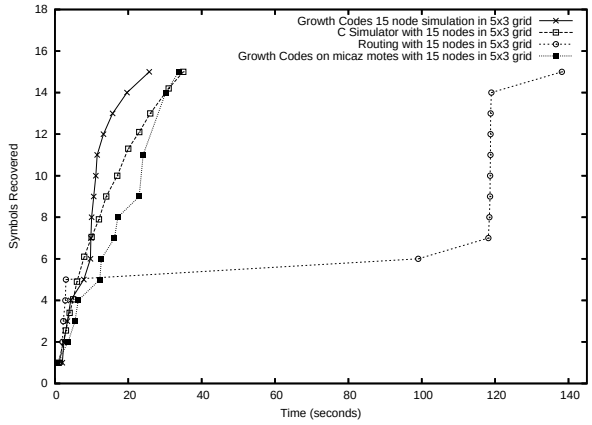

Figure 6: Data recovery on a 5x3 grid network. The sink node is in the middle of the grid

In this section, we introduce a C-simulator that we wrote to evaluate the performance of the various coding schemes. In addition, we use TinyOS [2] and the accompanying simulator TOSSIM to simulate a network of nodes with our implementation of the Growth Codes protocol. Finally, we use micaz motes from Crossbow [1] to test the protocol in a real setting. We also perform experiments to compare with routing, we use MultiHopRouter, a routing protocol included in TinyOS for mote-based sensor networks. We modify the Surge application (which uses MultihopRouter to setup and maintain a forwarding tree) to simulate a network of nodes which use this forwarding tree to route data towards the sink node.

Because no distributed encoding schemes exist for the Soliton distributions, we can only compare how Growth Codes achieve faster data dissemination to the sink node(s) compared to when no coding is used in the network.

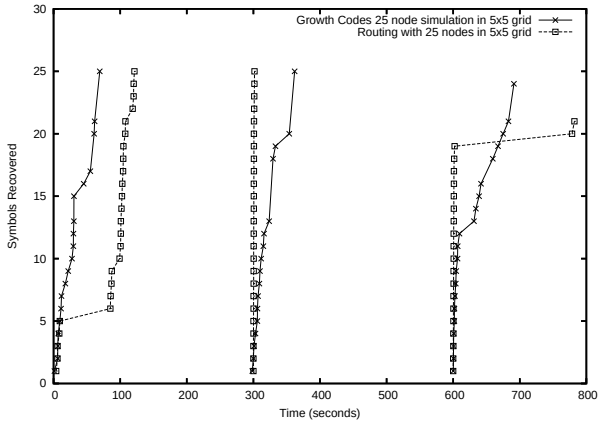

Figure 7: Data recovery in the event of node failures on a 5x5 grid network. The sink node is in the middle of the grid

Our purpose in presenting the results of the experiments and simulations is twofold. Firstly, we demonstrate that relaxing some of the assumptions of our abstract model does not result in a degradation of performance. Indeed, the simulation results of our round based C-simulator with the modeling abstractions present match closely with the results obtained with TOSSIM and the motes experiment. Thus, introducing real devices, a MAC protocol and nodes with limited memory and computational power does not distort the bigger picture of the performance of Growth Codes. After establishing the fidelity of our C simulator, we employ it to perform larger simulations in later sections, where we are not limited by the size of our physical testbed or the vagaries of the TOSSIM simulator. Secondly, our claims on the benefits of Growth Codes in a zero-configuration setting are substantiated.

In Figure 6, we see how the various schemes compare. The network consists of 15 nodes arranged in a 5x3 grid topology. The radio range is set such that a node can communicate only with its x- and y- axis neighbors (and not diagonally). The middle node on the grid is the sink. Using Growth Codes, each nodes sends one of its codeword symbols to a random neighbor every 1 second. We observe that both the TOSSIM and the C simulator as well as the micaz motes are able to transmit all data to the sink in about 40 seconds. With multihop routing, each node sends its data towards the sink every 1 second, but it takes a while for the forwarding tree to be established. If the next-hop for a node has not been established, the node broadcasts its data. We observe that the sink receives data from it 4 neighbors fairly quickly but after that there is a huge lag during which time the routing is being setup. This maybe an artifact of the particular implementation of MultiHopRouter used in TinyOS, but since this is the only suitable routing protocol available on TinyOS we use it for comparison purposes. We aim to compare with better routing protocols available on the TinyOS platform in future.

We now look at how node failures affect data recovery and how Growth Codes can recover from them. We use a 5x5 grid topology and assume that nodes generate new data every 300 seconds. Using routing, after the forwarding tree has been established, the nodes can get their data across very quickly. Later on, at t=600 seconds, when new data is generated one of the nodes close to the sink fails. At t=650, three more nodes which were routing through the already failed node also fail before they have a chance to re-establish their paths to the sink. This results in their data never reaching the sink and as can be observed from Figure 7, the curve

never reaches the maximum data recovery of 25. On the other hand, the Growth Codes allows data to be recovered from the three nodes mentioned above by disseminating their data to other neighboring nodes. While the experiment is designed to highlight the advantage of Growth Codes, such scenarios are likely to be common in large sensor networks where nodes fail with a fair degree of regularity. Growth Codes perform well with minimal reliance on transient components, e.g., optimized routes, that take a long time to recompute once broken. It is this ability that makes them particularly well suited for large-scale sensor networks.

Note that in the above experiments, we organize micaz motes into a grid topology by algorithmic means, i.e. by modifying the implementation so that each node is aware of and can communicate with only the nodes adjacent to it in the grid topology. Nodes physically arranged in a grid formation still show erratic radio connectivity and hence it is difficult to impose a grid topology by means of physical placement alone.

9. GROWTH CODES IN A DISTRIBUTED SETTING

In the following sections, we look at how fast Growth Codes can help the sink recover data in a completely distributed environment. We evaluate the efficacy of the coding technique both as a function of the underlying topology and as a function of the size of set of failed nodes. We compare with a random "diffusion" scheme where nodes randomly exchange data with their neighbors just as in Growth Codes but no coding is used.

In all simulations we use the round based C simulator and unless otherwise stated, we assume that a sensor node has a storage capacity $C = 10$, i.e., the node can store up to 10 codeword symbols at any given time. With each node, having a storage memory of $C = 10$, a maximum of $10N$ codewords can be stored in the network at any time. This is sufficient to facilitate good spread and mixing of symbols generated by the nodes. We also performed simulations with larger values of C, but the results were not noticeably different from what is presented below.

9.1 Growth Codes in a Random Topology

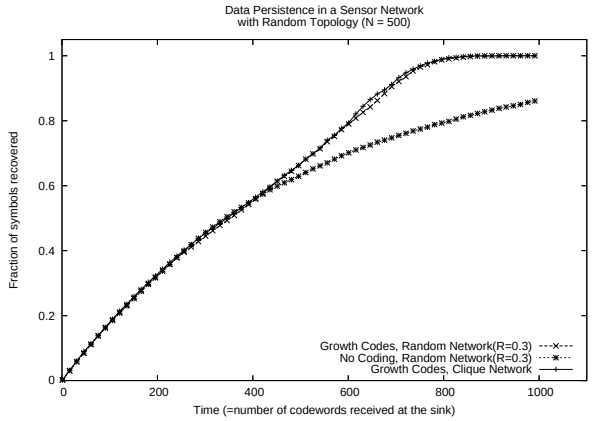

Figure 8: Data recovered at the sink in a 500 node random network of radius $R = 0.3$ as more and more codewords are received

In any practical deployment of sensor nodes in a geographical area, typically the network consists of wireless sensor nodes that have a certain range within which they can communicate. In such a scenario, the nodes aggregate in a natural topology defined by the wireless range of the nodes.

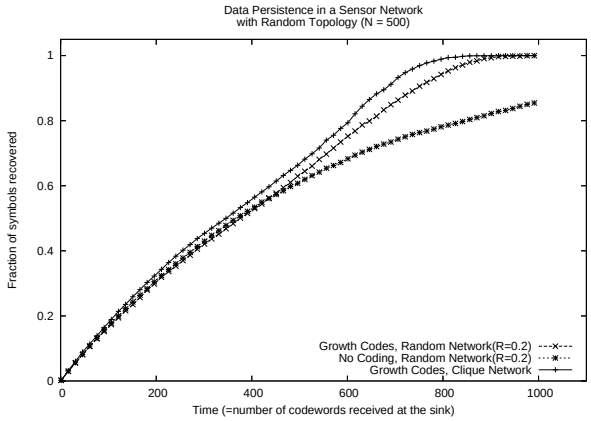

Figure 9: Data recovered at the sink in a 500 node random network of radius $R = 0.2$ as more and more codewords are received

We simulate this network as a 1x1 square, by placing sensor nodes uniformly at random within the square. A pair of nodes, is connected by a link if they are within a distance R from one another. The parameter R is called the *radius* of the network. This is similar to the connectivity model used within [28].

We assume each node in the network initially generates a symbol x_i. There is one sink that is attached to a single random node in the network. Hence it is able to receive one codeword symbol in each round. Therefore, the round number is the same as the number of codewords received at the sink. The nodes follow the encoding and data exchange protocol based on Growth Codes as described in Section 4.1.

We compare with the case when the network nodes do not encode any data but instead exchange the original symbols. To facilitate a fair comparison, when not using any coding, the sensor nodes behave in exactly the same way as when they use Growth Codes except that they do not transition to degrees higher than 1. Hence all the codewords generated are of degree 1.

We now compare how in a randomly formed network where the density of nodes and links will vary from region to region, Growth Codes can be used to deliver data at a fast rate to the sink node. We consider a 500 node network. The sink is attached to a random node in the network.

Figure 8 depicts the fraction of data recovered by the sink as a function of the number of received codewords in a random network with radius of $R = 0.3$. The results are an average of 100 simulation runs. On the X-axis, the number of codewords received by the sink are plotted while the Y-axis is the fraction of symbols recovered from the received codewords.

A random network with a radius of $R = 0.3$ is fairly well connected. For comparison purposes, we plot the fraction of data recovered as a function of number of codewords received in a network with clique topology. The performance of Growth Codes is roughly the same in both topologies.

We now evaluate the data recovery rate using Growth Codes when the network has a radius of $R = 0.2$ implying that the network is much sparser. From Figure 9 we can observe that the data delivery rate of Growth Codes does not deteriorate very much even in a much sparser random network.

We observe that when the number of received codewords is small,

the two protocols recover data at the same rate (since in the beginning, Growth Codes produce only degree 1 codewords). On the other hand when a substantial number of codewords have been received, the Growth Codes protocol can achieve much faster recovery rates. Using Growth Codes, the sink is able to recover *all* symbols much earlier than when no coding is used.

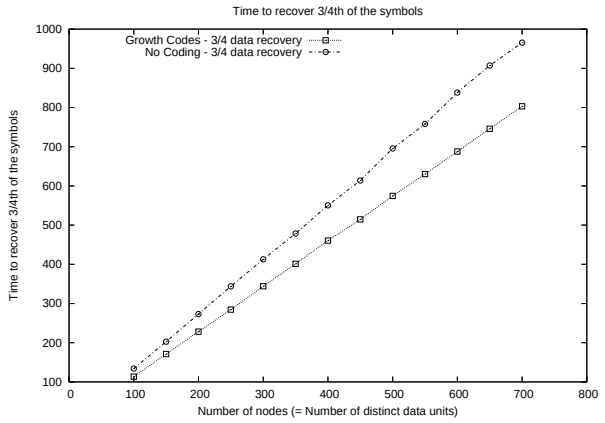

Figure 10: Time taken to recover 75% of the data in a 500 node network when the nodes use Growth Codes versus no coding

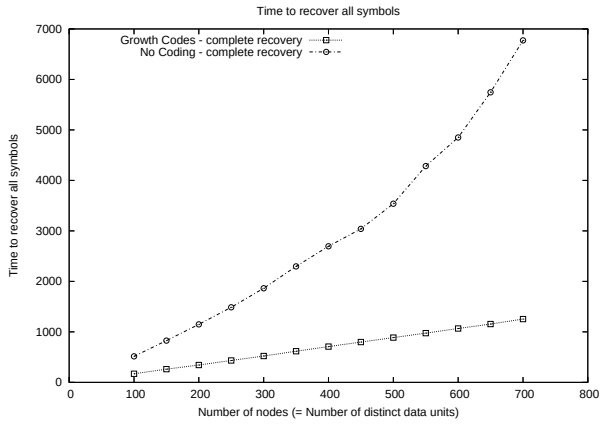

Figure 11: Time taken to recover all the data in a 500 node network when the nodes use Growth Codes versus no coding

Figure 10 depicts how many rounds on average are required before the sink is able to recover 75% of the original N symbols. On the X-axis, we vary the number of sensor nodes in the network which is the same as the total number of symbols (since each sensor node generates one symbol). On the Y-axis, we measure the time taken to recover 75% of the symbols. Figure 11 depicts the time taken to recover *all* the symbols.

To recover 75% of the data, both protocols require the number of codewords to be linear in the number of nodes N with the Growth Codes protocol requiring a smaller amount. Recovering *all* the symbols using Growth Codes still appears linear in the number of nodes, however, without coding, the number of codewords required show a clear non-linear increase. Since the unencoded retrieval process is closely related to the classical *Coupon Collector's Problem*, it is not surprising to see the unencoded version require a long time to acquire the last few symbols.

9.2 Growth Codes in Disaster-Prone Networks

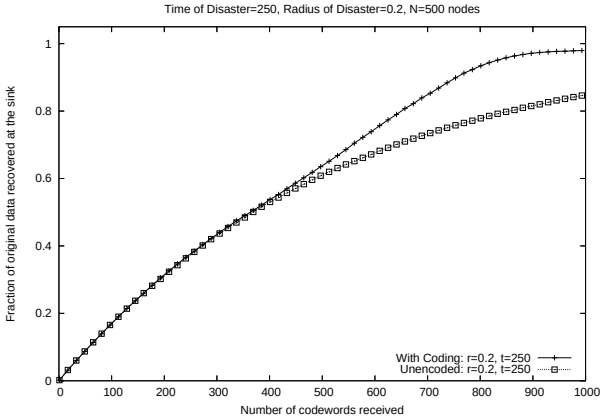

Figure 12: Data recovered at the sink in a 500 node random network in a disaster scenario of impact radius $r = 0.2$ at time $t = 250$

Growth codes are particularly effective for information collection in disaster-prone areas where the whole network or parts of it may be destroyed or affected in some way. For example, in the event of a flood, a subset of the sensor nodes may get submerged and stop functioning. In the case of an earthquake, a whole region containing sensor nodes may get destroyed. Typically, the effect of the disaster on the sensor network will show strong spatial correlation in the sense that nodes close to one another are more likely to either survive together or get destroyed together.

We simulate a disaster in a sensor network by disabling part of the network at the time of the disaster. We define r such that at the time of the disaster, all nodes within r distance of the center of the impact are disabled. This in effect, changes the topology of the network since the disabled nodes and all links connecting them become non-functional.

We now discuss how the time of disaster and the radius of impact affects the network's ability to deliver data to the sink. Figure 12 shows how much data can be recovered at the sink if a disaster of impact radius $r = 0.2$ disables part of the network at time $t = 250$. All nodes within a radius of $r = 0.2$ from the center of the impact are disabled. This amounts to about $1/8^{th}$ of the total nodes. Nevertheless, the data recovery rate at the sink when the nodes use Growth Codes to encode and distribute data is not severely affected.

If the disaster occurs soon after the data generation by the sensor nodes, the nodes that eventually get disabled may not have a chance to spread their data outside the region of impact and that data may get lost. Even if the data from disabled nodes does get to the surviving nodes, the distribution of data from disabled nodes and the data from surviving nodes may become skewed. If the nodes are not using any coding scheme, this skew will affect the data recovery rate at the sink. On the other hand, if the nodes are using codes such as Growth Codes, the skew will not matter since high degree codewords containing data from disabled nodes will be constructed.

In Figure 13, we depict how much data can be recovered by the various protocols and how long does it take to recover that data in case a disaster disables a part of a 500 node network soon after data generation by the nodes. On the X-axis, we vary the impact radius of the disaster. The number of disabled nodes is roughly proportional to the square of the impact radius r. On the Y-axis, we display how long it took for the sink to recover all symbols that sur-

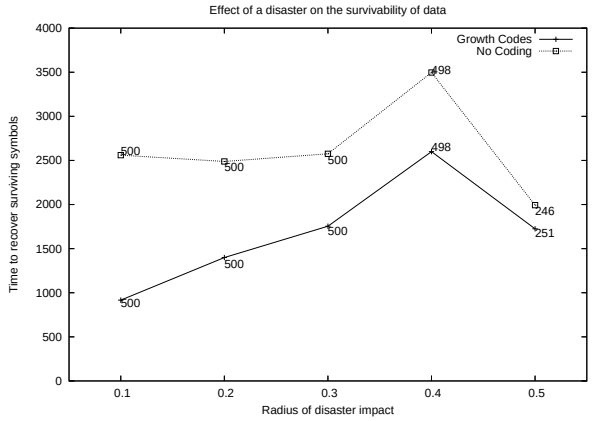

Figure 13: Time required to recover surviving data for a disaster at time $t = 50$ of varying intensities. The numbers next to the points indicate the number of symbols recovered in that experiment.

vived the disaster. The number of symbols recovered is indicated next to the plotted point. Note that it takes less time to recover all recoverable symbols for the disaster of size 0.5 because a much smaller fraction of symbols is recovered. We can observe that if the disaster radius is smaller than $r = 0.4$ (that covers approximately half the nodes), most of the symbols do survive the disaster. Only when a substantial portion of the network is destroyed, a number of symbols are lost since a lot of nodes as well as their storage memory is destroyed. Yet, by using Growth Codes, the surviving data can be recovered reasonably fast by the sink.

10. NODES THAT PERIODICALLY GENERATE NEW DATA

To this point, we have considered a network setting where nodes take a single measurement of their environment, the goal being to deliver as much of that data as possible to the sink(s). We now extend Growth Codes to an environment where nodes periodically sample the environment. We assume that with period G, nodes generate data of size s_d. We call the data generated by a node in successive time periods as belonging to different successive *generations*. The first data generated by a sensor node is of generation 1.

We assume a fixed storage size of C at each node. The storage must now be shared across multiple generations. Typical sensor nodes have a limit on the maximum packet size S they can send or receive. We partition the memory of each node of size C into chunks of size S so that there are $\lceil C/S \rceil$ chunks.

10.1 Clustering Across Generations

One concern when using Growth Codes across multiple generations is that the size of the coefficient describing the codeword could grow large for large-degree codewords. Let s_d be the amount of space required for storing the data and s_c for storing the *coefficient*. Let γ be the number of generations we wish to store in a chunk of S. This leaves on average size S/γ for each generation, and a naive storage scheme would require that γ be chosen such that $S/g_m \geq s_d + s_c$. Here g_m indicates the number of generations that comprise a cluster.

To reduce the coefficient overhead across generations, we introduce the notion of *clustering*. A cluster is a set of codewords across several generations. The codewords have the same coefficients (i.e., the origins of the symbols used that comprise the codeword are the same), but the data in each codeword is from a different generation.

Cluster 1	Cluster 2	Cluster 3	Cluster 4
Coeff	Coeff	Coeff	Coeff
Gen 3 codeword	Gen 4 codeword	Gen 7 codeword	Gen 10 codeword
	Gen 5 codeword	Gen 8 codeword	Gen 11 codeword
	Gen 6 codeword	Gen 9 codeword	

Figure 14: Storing Clusters in Memory.

An example of how memory is used with clustering is depicted in Figure 14 in which the number of generations per cluster, $g_m = 3$. A cluster's smallest numbered (i.e., earliest) generation is numbered $1 \pmod 3$, and the largest numbered (i.e., latest) generation is numbered $0 \pmod 3$. In this example, there is sufficient memory to store 9 codewords and 4 coefficients (i.e., $S \geq 9s_d + 4s_c$). To make room for the codewords cluster 4 (generation 10 and 11), codewords for generation 1 and 2 were removed. When generation 12's codeword is generated, its storage will necessitate the eviction of codeword 3. Hence, cluster 1 can be completely removed at that time. When generation 13 arrives, cluster 5 is initially formed, and codeword 4 will be removed at that time (but generations 5 and 6, and hence the coefficient of cluster 3, will remain).

Since all codewords within a cluster have the same coefficient, the codeword cannot be modified (i.e., "grown" to a higher degree) until the cluster is fully formed (e.g., in the example above, the codewords for generation 10 and 11 must remain as the original data symbols generated by that node for those generations until generation 12's data is also available). This delay is not a significant concern since we know that the optimal growth rate of Growth Codes stays fixed at degree one for a considerable amount of time.

Since all generations in a cluster share the same coefficient, the point in time at which the codewords grow to the next degree is the same. Hence, the time will either be too soon for the most recent generation in a cluster, or too late for the oldest generation in a cluster. Since our approach is to remain conservative about when to transition to the next degree, we recommend the use of the transition time of the most recent generation in the cluster.

The above discussion raises an issue about how many generations of data can be encoded in a single packet. Also, transmission channels have typically have a small limit on the packet size (Maximum Transmission Unit) which can be sent over the channel. In case a packet containing multiple generations is bigger than the MTU of the channel, a simple fragmentation and reassembly can be added to the implementation to take care of this issue. We do not explore this issue further in this paper due to space constraints.

10.2 Optimal Cluster Size

As cluster size is increased, the size of the coefficient per generation decreases. If the space allocated to each generation is fixed, then by decreasing the overhead of the coefficients, data can reside longer in memory. However, since codewords within a cluster cannot be grown until all generations are available for that cluster, a larger cluster size reduces the time over which a codeword can grow.

Let g_m be the maximum number of generations in a cluster. It can be shown that the optimal value of g_m is bounded from below by $g_m = \dfrac{\sqrt{2Ss_c} - s_c}{s_d}$. If memory size is S, the coefficient size s_c,

the codeword size is s_d and the maximum generations in a cluster can be g_m, then the maximum size of a cluster is $s_c + g_m s_d$. The number of clusters in memory is approximately $S/(s_c + g_m s_d)$. The number of codewords of successive generations in memory is then $g_m S/(s_c + g_m s_d)$. Since a new generation codeword is added after time G (and the oldest codeword removed at the same time), the time for which a generation codeword stays in memory is given by $T_g = g_m SG/(s_c + g_m s_d)$.

It can easily be observed that the generation's codeword in the cluster must remain fixed for $g_m - 1$ generations, the second oldest generation's codeword for $g_m - 2$ generations and so on. Hence, an "average" codeword must be fixed for $\frac{g_m - 1}{2}$ generations, and the time for which a typical codeword is fixed is therefore $G\frac{g_m - 1}{2}$.

The time over which a codeword grows uninterrupted according to the Growth Codes protocol is then given by $T_{growth} = T_g - G\frac{g_m - 1}{2}$. The optimal value of g_m is one that maximizes the time over which an average codeword can grow. This is given by $g_m = \frac{\sqrt{2Ss_c} - s_c}{s_d}$, where we omit the derivation due to lack of space.

T_g, which is the time codewords of a particular generation stay in memory, increases monotonically with g_m. But the time for which a codeword grows uninterrupted according to the Growth Codes protocol is given by T_{growth} which peaks at the optimal value of g_m . For values of g_m higher than this, T_{growth} decreases but the extra low degree codewords generated by the constant resetting of the clusters also grows, resulting in a slight loss of efficiency.

11. CONCLUSIONS

The goal of this paper is to investigate techniques to increase the persistence of sensor networks. We propose Growth Codes, a new class of network codes particularly suited to sensor networks where data collection is distributed. Unlike previous coding schemes, Growth Codes employ a dynamically changing codeword degree distribution that delivers data at a much faster rate to network data sinks. Furthermore, the codewords are designed such that the sink is able to decode a substantial number of the received codewords at any stage. This is particularly useful in sensor networks deployed in disaster scenarios such as floods, fires etc. where parts of the network may get destroyed at any time.

Our simulations and experiments demonstrate that Growth Codes outperform other proposed methods in settings where nodes are highly prone to failures. While persistence in sensor networks was one application we studied, we believe Growth Codes have wider applicability. The critical insight behind Growth Codes is to intelligently preserve "memory" of a network, making it more robust and resilient. The idea is potentially very useful in other dynamically changing environments like file swarming p2p frameworks, and we are actively exploring other similar applications of Growth Codes.

12. REFERENCES

[1] Crossbow micaz motes.

[2] TinyOS Homepage.

[3] S. Katti, D. Katabi, W. Hu, H. Rahul and M. Medard. The Importance of Being Opportunistic: Practical Network Coding For Wireless Environments. In *Allerton Annual Conference on Communication, Control and Computing*, 2005.

[4] A. Albanese, J. Blömer, J. Edmonds, M. Luby, M. Sudan. Priority Encoding Transmission. In *IEEE Transactions on Information Theory*, volume 42, 1996.

[5] A. G. Dimakis, V. Prabhakaran and K. Ramchandran. Ubiquitous Acess to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes. In *Information Processing in Sensor Networks*, 2005.

[6] A. Kamra, J. Feldman, V. Misra and D. Rubenstein. Data Persistence for Zero-Configuration Sensor Networks. In *Technical Report, Department of Computer Science, Columbia University*, Feb. 2006.

[7] A. Manjeshwar and D.P. Agrawal. Teen: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks. In *Parallel and Distributed Processing Symposium*, 2001.

[8] A. Scaglione and S. D. Servetto. On the interdependence of routing and data compression in multi-hop sensor networks. In *ACM Conference on Mobile Computing and Networking*, 2002.

[9] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proceedings of INFOCOM*, 2005.

[10] C. Intanagonwiwat, R. Govindan and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *ACM Conference on Mobile Computing and Networking*, 2000.

[11] C. Y. Wan, S. B. Eisenman, A. T. Campbell, J. Crowcroft. Siphon: Overload Traffic Management using Multi-Radio Virtual Sinks. In *ACM Conference on Embedded Networked Sensor Systems*, 2005.

[12] D. Marco, D. Neuhoff. Reliability vs. Efficiency in Distributed Source Coding for Field-Gathering. In *Information Processing in Sensor Networks*, 2004.

[13] I. F. Akyildiz, M. C. Vuran, O. B. Akan. On Exploiting Spatial and Temporal Correlation in Wireless Sensor Networks. In *Proceedings of WiOpt 2004: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 71–80, Mar. 2004.

[14] J. Byers, J. Considine, M. Mitzenmacher and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *Proceedings of SIGCOMM*, 2002.

[15] J. Considine. Generating good degree distributions for sparse parity check codes using oracles. In *Technical Report, BUCS-TR 2001-019, Boston University*, 2001.

[16] J. W. Byers, M. Luby, M. Mitzenmacher, A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proceedings of SIGCOMM*, 1998.

[17] Lin and Costello. *Error Control Coding: Fundamentals and Applications*. 1983.

[18] M. Luby. LT Codes. In *Symposium on Foundations of Computer Science*, 2002.

[19] M. Li, D. Ganesan and P. Shenoy. PRESTO: Feedback-Driven Data Management in Sensor Networks. In *ACM/USENIX Symposium on Networked Systems Design and Implementation*, 2006.

[20] M. Luby, M. Mitzenmacher, M. A. Shokrollahi and D. Spielman. Efficient Erasure Correcting Codes. In *IEEE Transactions on Information Theory*, volume 47, pages 569–584, 2001.

[21] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance. In *Journal of the ACM*, volume 36, pages 335–348, 1989.

[22] M. Perillo, Z. Ignjatovic and W. Heinzelman. An Energy Conservation Method for Wireless Sensor Networks Employing a Blue Noise Spatial Sampling Technique. In *Information Processing in Sensor Networks*, pages 116–123, 2003.

[23] M. Sartipi and F. Fekri. Source and Channel Coding in Wireless Sensor Networks using LDPC Codes. In *EEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[24] N. Cai, S. Y. R. Li and R. W. Yeung. Linear Network Coding. In *IEEE Transactions on Information Theory*, volume 49, pages 371–381, 2003.

[25] P. Desnoyers, D. Ganesan and P. Shenoy. TSAR: A Two Tier Storage Architecture Using Interval Skip Graphs. In *ACM Conference on Embedded Networked Sensor Systems*, 2005.

[26] P. J. M. Havinga, G. J. M. Smit and M. Bos. Energy Efficient Adaptive Wireless Network Design. In *The Fifth Symposium on Computers and Communications*, 2000.

[27] R. Ahlswede, N. Cai, S. Y. R. Li and R. W. Yeung. Network Information Flow. In *IEEE Transactions on Information Theory*, volume 46, pages 1004–1016, 2000.

[28] R. Chandra, C. Fetzer and K. Hogstedt. A Mesh-based Robust Topology Discovery Algorithm for Hybrid Wireless Networks. In *Proceedings of AD-HOC Networks and Wireless*, Sept. 2002.

[29] R. Koetter and M. Medard. An Algebraic Approach to Network Coding. In *ACM/IEEE Transactions on Networking*, volume 11, pages 782–795, 2003.

[30] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge International Series on Parallel Computation.

[31] S. Acedanski, S. Deb, M. Medard and R. Koetter. How Good is Random Linear Coding Based Distributed Networked Storage. In *Workshop on Network Coding, Theory and Applications*, 2005.

[32] S. Pattem, B. Krishnamachari and R. Govindan. The Impact of Spatial Correlation on Routing with Compression in Wireless Sensor Networks. In *Information Processing in Sensor Networks*, pages 28–35, 2004.

[33] S. S. Pradhan, J. Kusuma and K. Ramchandran. Distributed compression in a dense microsensor network. In *IEEE Signal Processing Magazine*, volume 19, pages 51–60, Mar. 2002.

[34] T. Arici, B. Gedik, Y. Altunbasak and L. Liu. PINCO: a Pipelined In-Network COmpression Scheme for Data Collection in Wireless Sensor Networks. In *International Conference on Computer Communications and Networks*, 2003.

[35] T. Ho, M. Médard, M. Effros and D. Karger. On Randomized Network Coding. In *Allerton Annual Conference on Communication, Control and Computing*, Oct. 2003.

[36] W. Heinzelman, A. Chandrakasan and H. Balakrishnan. Energy-Efficient Communication Protocols for Wireless Microsensor Networks. In *Hawaii International Conference on Systems Sciences*, 2000.