Data Persistence for Zero-Configuration Sensor Networks

Abhinav Kamra*, Jon Feldman‡, Vishal Misra* and Dan Rubenstein*

*Department of Computer Science

Columbia University, New York

Email: {kamra, misra, danr}@cs.columbia.edu

‡Google Inc.

Email: jonfeld@google.com

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 via simulations that in typical sensor network topologies, our novel protocol can preserve about 10-20% more data than when no coding is done. We also design a dynamically changing codeword degree distribution based on our results and show that it delivers data at a much faster rate compared to other well known degree distributions such as Soliton and Robust-Soliton.

I. 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*, 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 [1] or routing with re-coding are applied [2], there can be a significant delay between the time a unit of data is generated and the time at which it reaches the delivery 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. Our solutions are based on the observation that even though there is limited bandwidth to forward data toward 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.

Our solutions are designed specifically for "zero-configuration" networks: networks whose data collection and transmission must be initiated immediately, before nodes have the 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 or direction of a sink, or the complete sensor network topology are unknown. Disaster situations are likely to benefit from a zero-configuration design, as sensors may be dropped into the critical

region to monitor an ongoing disaster, or the disaster itself may disrupt a planned network architecture. In both of these cases, there is little to no knowledge of the current conditions, and any advanced configuration is likely to be obsolete.

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 encoded 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 unreceived 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 mathematically 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 distribution, we compare the performance of a degree distribution based on Growth Codes with other well known distributions such as *Soliton* and *Robust Soliton* [3]. Furthermore, we perform simulations in 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.

The rest of the paper is organized as follows: The next section details some previous related work. In Section III, we formulate the problem and describe the network setting in which it is employed. In Section IV we describe the various solution approaches including the novel approach based on Growth Codes. We also describe the encoding/decoding algorithms used. In

Section V we mathematically analyze the encoding and distribution protocol based on Growth Codes and prove interesting properties of the said protocol. In Sections VI and VII we present simulation results to study the performance of our protocol in various sensor network settings. We conclude in Section IX.

II. Related Work

In distributed storage networks, it has been shown [4] that it is beneficial to store encoded symbols of symbols instead of the original data. Dimakis et. al. use erasure codes [5] to store a file in a distributed setting so that it can be downloaded from a limited number of available sources. Yeung et. al. [6] also show that in a multicast scenario, it is beneficial to encode the data in order to maximize the delivery rate to the destination. They further show [7] that Linear Coding suffices. That is, it is sufficient to store linear combinations of the data as the encoded symbols.

Traditional error-correcting erasure codes can also be used to achieve the goal of encoding data such that if some of the encoding symbols are lost, data can still be recovered. Reed-Solomon codes [8] are optimal block erasure codes that have been traditionally used for error correction but can be used for erasure-resilient data transfer also [9]. But it is impractical to use these codes in a distributed sensor network environment since the encoding/decoding complexity of these codes is high, requiring substantial time at intermediate nodes to implement the coding procedure.

The benefits of sending combinations of data instead of original data in the context of content distribution networks has also been studied in various works [10], [11]. Luby et. al. propose using *Priority Encoding Transmission* [12] to send data over lossy channels. The priority value of each part of the data determines the fraction of encoded packets sufficient to recover that part. The codewords are based on a class of erasure-resilient codes which have high encoding/decoding complexity. Furthermore, it is difficult to use these codes in an environment where the data is completely distributed to begin with, such as in a sensor network where each node collects its own data.

Byers et. al. use erasure-resilient codes to optimize large transfers across adaptive overlay networks [13]. They also propose collaboration between network nodes which have some non-overlapping portions of the data to exchange *re-coded* data with each other. The protocol involves

exchanging control messages between peers such as approximate reconciliation trees to filter out duplicate data. Such recoding might be a useful augmentation to Growth Codes. However, previous work does not address the challenges of providing feedback to an overwhelmingly large number of nodes that exist in sensor networks, and we leave the design of a recoding scheme for Growth Codes as future work.

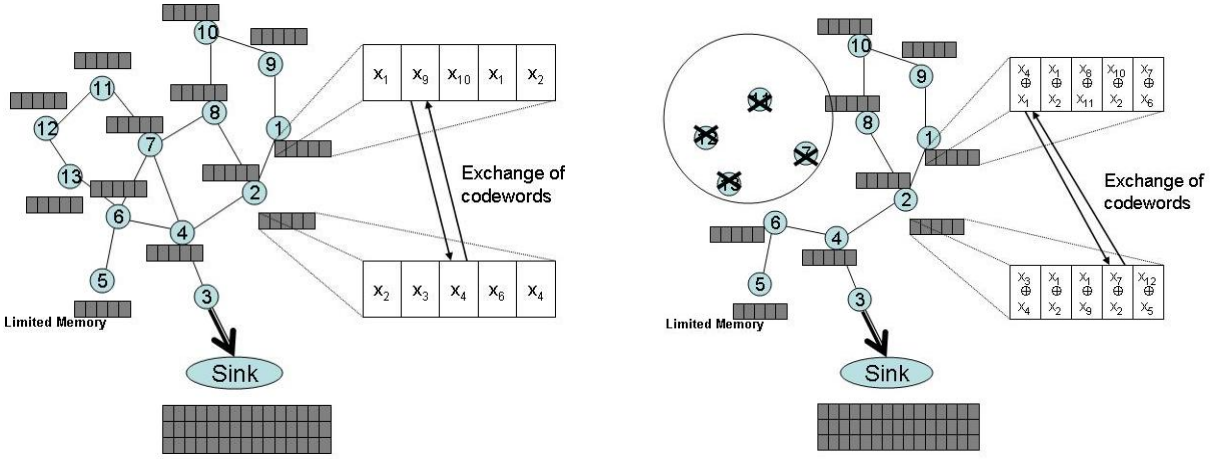
Luby et. al. describe a class of erasure codes known as *Digital Fountain* [14] to optimize large transfers across a network. Tornado codes [15] and more recently LT codes [3] are examples of such codes. To recover N symbols at the destination, they require the reception of slightly more than N encoded symbols but have much faster encoding/decoding. These codes focus on the problem of choosing an encoding such that *all* the data can be recovered using a minimum number of codeword symbols.

Most of the encoding techniques described above decrease the expected time of delivery of *all* data to the sink. All assume that the data to be encoded is located at a centralized point. In sensor networks, having such a point other than the sink is highly unlikely, since it is impractical to try to aggregate all or a substantial subset of the data at one point for encoding due to low storage memory of nodes and limited bandwidth restrictions. An encoding scheme which adapts well in a zero-configuration environment and addresses these specific constraints is imperative.

III. Problem Setup

Our formulation contains 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. Each sensor node periodically senses the data in its immediate neighborhood with the intention of forwarding this data to a sink for further processing. The network exhibits a funneling effect exists such that the rate at which the nodes as an aggregate generate data greatly exceeds the communication capacity of the network. Hence, data becomes “backed up”, and must reside temporarily in the nodes as it waits its turn to be 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 also have some computational power to manipulate data in transit, i.e., they can compress or recode data to increase delivery efficiency [1], [16].

There are also some specifics of our formulation that distinguish our environment from much



(a) Snapshot of the network at time t_1

(b) Snapshot of the network at time t_2

Fig. 1. Snapshot of a sample sensor network at time t_1 and a later time t_2. Some nodes failed during the intervening time

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 before the disaster, the disaster itself may significantly alter the network configuration, making the pre-planned configuration obsolete. These specifics translate to our problem as follows:

- Our network is "zero-configuration", such that nodes must operate knowing very limited general global properties, such as the (approximate) number of nodes deployed and the rate at which the sinks should be able to collect data. In addition, nodes can quickly "learn", once deployed, about their immediate sensing environment: the only topology information we assume a node has is its set of neighboring nodes with whom it can communicate directly.
- 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 [17], [18] and optimized routing [19], [20] are secondary issues and are not directly addressed in this paper.

A. Sensor Networks that Need Persistence

Figure 1(a) depicts our view of the sensor network at some time t_1. 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.

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.

Figure 1(b) shows the same network as in Figure 1(a) but at a later time $t_2 > t_1$, after a set of nodes in a circular region has suddenly failed (e.g., a structural collapse in an earthquake). The failure of nodes in the 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*.

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.

B. 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). Each sensor node collects information from its environment and generates data at some rate λ_1. A sink node absorbs data at a different rate λ_2, where λ_2 depends on the number of sensor nodes to which the sink connects. In the network, the neighboring nodes can exchange data at a third rate λ_3.

Since we are making a preliminary foray into the problem of persistence, we believe it is most appropriate to thoroughly analyze a preliminary 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 we expect the protocol can easily be extended in order to relax these assumptions, and we touch upon these points in Section VIII:

- Node configurations are homogeneous: each node has a similar memory size C.
- Nodes generate new data items periodically (e.g., one can assume some fixed period $\tau = \frac{1}{\lambda_1}$) and each data is timestamped according to its *epoch*. Epochs are assumed to be large enough such that clock skew and synchronization issues prevalent in sensor networks [21], [22] do not affect a nodes' ability to determine whether a data item generated at another node A occurred in a previous, same, or subsequent epoch as the data generated in another node B. In other words, each data item has a globally consistent epoch timestamp.
- 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 [23], [24] as well as the likelihood that some data generated is more important than others (e.g., at nodes closer to regions of anomalous activity).
- Time is divided into *rounds*, with multiple rounds residing within each epoch. In each round, every node selects a neighbor and these neighbors can exchange information. Note that a node can have multiple exchanges within a round (if selected by several neighbors), but that the number is expected to be fairly small (i.e., the number of neighbors who choose the node to communicate plus one). We emphasize that the division of time into rounds is merely to facilitate the description and evaluation of our techniques. The described techniques are easily extended to a non-discretized environment.

Figure 2 shows a sample time line of various events in the network from the point of view of a single node. Initially, in every round, the node exchanges degree 1 codewords with one of its neighbors chosen at random. At the first transition point, the node switches to degree 2 codewords. After that degree 2 codewords are exchanged. At *Epoch 2*, the node generates the next data and the cycle starts again.

We believe that once we have a solid understanding within this simplified model, it will significantly facilitate generalizing the persistence techniques to environments where these assumptions

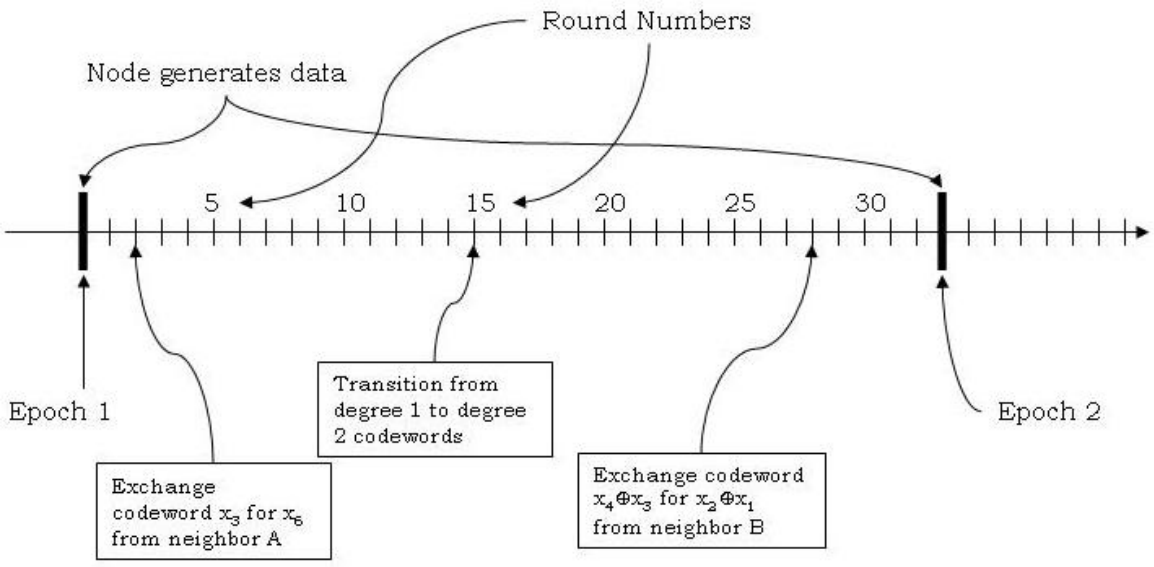


Fig. 2. Sample time line of various events in the network from the point of view of a single node

are relaxed.

IV. 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. 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 each round, each node must decide what information it must transmit to its neighbor, and what and how the incoming information from its neighboring nodes is stored.
- In each round, each sink collects new codeword symbols and must *decode* this information to retrieve the original data. We will examine several decoding procedures.

A typical node in the network, when communicating with one of its neighbors, has to look at the data it has in the limited storage and encode it in some way. The neighbor does the same with its data and then the two nodes exchange these encoded symbols. The idea being that as more and more nodes exchange these codeword symbols, data generated by all nodes will be well distributed throughout the network. The nodes can use various encoding methods to achieve this effect:

- **No Coding:** Nodes can simply exchange the original data units they generated. In the beginning (after initial deployment in the sensing region), each node can store the data it generated in all C of its storage locations. In the case where the nodes generate data periodically, data from previous generations is either removed immediately or slowly phased out when new data is generated. In a zero-configuration setting where all data has equal priority, there is no preference of data to forward or the direction in which it should be forwarded. Hence, when communicating with a neighbor, each node picks a random symbol from its set of stored symbols to exchange and a random neighbor with which it makes the exchange. In this way, the original data symbols will soon be fairly well spread in the network.

  The number of symbols that can be recovered by the sink may be much less than the number of symbols received, since the sink will likely receive many duplicates. In the well studied *Coupon Collectors' Problem* [25], 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 distinct original symbols.
- **Random Linear codes [26]:** 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 II, they are impractical in our sensor network setting.
- **Erasure based codes:** Nodes can store and exchange codeword symbols formed using erasure codes. These can either be optimal erasure codes such as Reed-Solomon [8] or erasure codes based on sparse bipartite graphs such as Tornado [15] or LT codes [3] 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.

  These codes are XOR based codes, i.e. each codeword symbol is an XOR of a subset of the N original symbols. The number of symbols which are XOR-ed together to form the codeword is referred to as the *degree* of the codeword. If each original symbol is larger than a bit, then they are XOR-ed in a bitwise fashion to get a codeword of the same size.

These codes choose the degree of each codeword to fit a pre-determined *degree distribution*.. For example, if the degree distribution is given as $\pi_{i\in[1,N]}$ values then $\forall i, 0 \leq \pi_i \leq 1$ and $\sum_{i=1}^{N} \pi_i = 1$.

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) 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 which is defined as

$$\pi_1 = \frac{1}{N}, \forall i \in [2, N], \pi_i = \frac{1}{i(i-1)} \tag{1}$$

If the codewords are constructed according to the Soliton degree distribution, then the sink only needs N codeword symbols in expectation to recover all N original 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 note that these codes were not specifically designed to maximize the amount of information recovered by the sink when only a small number of codewords can be delivered.

It is also not clear how, in a distributed sensor network setting, the nodes can construct codewords that fit the desired degree distribution. For example, to construct a codeword symbol of degree 100, the node needs to have simultaneous access to all 100 original symbols to XOR them together into a codeword. This is highly infeasible in a completely distributed environment such as the one we are dealing with, where all the data is not initially aggregated at a single point.

A. Growth Codes

We now describe a novel technique to encode information in a completely distributed manner in a sensor network and deliver it to a sink even when the data is generated at different nodes. Our simulation results later in the paper will demonstrate that the technique works well even

when the nodes are unaware of the general topology of the network or the location of the sink node(s). The technique ensures that the sink is able to recover *all* N of the data from only a little more than N codeword symbols. Furthermore, even when only a small number of codeword symbols are received at the sink, a substantial amount of data can still be recovered.

We motivate the development of Growth Codes using the following toy problem. A sink is attempting to collect data over a series of rounds. The data can be XOR'd together to generate fixed-size codewords. The sink can select the degree of the arriving codeword, but the set of symbols that form the codeword are selected uniformly at random. 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 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.

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.

We introduce *Growth codes* and an associated encoding and data exchange protocol which is easily implementable in a fully distributed sensor network setting and is also highly efficient. 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 1 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-ed 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.

Definition 4.2: Given a set A of codeword symbols $s_1, s_2, \ldots, s_k$, the offline iterative decoder *D* works as follows: Initially the set X of recovered symbols is empty.

1) Decode all degree 1 codeword symbols and add them to set X and remove from set A. This is trivial since the degree 1 codewords are the same as the original symbols
2) From the remaining symbols in set A, choose a symbol s such that $dist(s, X)$ is the minimum.
3) If $dist(s, X) = 0$, throw away this symbol as a redundant or duplicate symbol.
4) If $dist(s, X) = 1$, decode a new symbol and add it to X. Goto Step 2.
5) If $dist(s, X) > 1$, then stop.

The remaining symbols have distance greater than 1 from X. More sophisticated techniques (e.g., gaussian elimination) could be used to extract additional data.

Table I describes in detail the encoding and data exchange protocol based on Growth Codes. 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.

Nodes that implement the encoding process receive data that belongs within a certain epoch. A sequence of increasing values, $K_1, K_2, \ldots, K_N$ 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 epoch in which the data was generated) where such data should be encoded in codewords of

degree no less than i. Hence, 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-ed into the codeword and its degree increased.

In the next section, we approach the problem mathematically to identify good theoretical values for the K_i.

V. Designing 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. Furthermore, this assumption holds when the network is a clique.

Solving for the optimal switching points is also very difficult for the decoder 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 could be used later on as further codewords arrive and additional data symbols are decoded. Hence, we will consider a pair of restricted decoders that throw away any codewords they receive that cannot be decoded immediately, and do not maintain them for later use.

A. Restricted Decoders

We define two more decoding algorithms that are less powerful than the decoder D. One would never use these decoders in practice, but they are used here because they are simpler to analyze and will help us prove bounds on the switching times K_i of Growth Codes.

TABLE I

ROUND-BASED VIEW OF ENCODING AND DATA EXCHANGE PROTOCOL BASED ON GROWTH CODES

Network Setup:

- N nodes connected in an arbitrary topology
- Each node can store up to C number of XOR codewords
- In each round, each sensor node exchanges one codeword symbol with one of its neighbors
- One or more sink nodes. Each sink node is attached to a random sensor node.
- In each round, a sink node receives the latest codeword received by the node it is attached to

Node i does the following:

- At an epoch, collect data from the environment and generate symbol x_i.
- Store x_i in all C storage locations available. Also, store x_i in a small separate storage which is not exchanged.
- Set $maxdegree = 1$
- In the t^{th} round after the epoch, do the following:
  1) Let $x =$ a randomly chosen codeword from one of the C storage locations.
  2) If $(degree(x) < maxdegree$ AND $x_i \notin x)$, then let $x = x \oplus x_i$
  3) While $(t \geq K_{maxdegree})$, $maxdegree++$;
  4) Pick a random neighbor and exchange codeword x for codeword y from the neighbor
  5) Store codeword y in the original location of x

Sink does the following:

- At an epoch, let X and Y be empty sets. X is the set of as yet recovered symbols and Y is the set of codewords not yet decoded
- In the t^{th} round after an epoch, do the following:
  1) Receive a codeword s from the node it is attached to. Suppose $degree(s) = d$
  2) If $dist(s, X) = 0$, throw away s as a redundant symbol
  3) If $dist(s, X) = 1$, recover a new symbol x_i by subtracting the already known $d - 1$ components from s. $X = X \cup s$
     - Look at the codewords in set Y. If $dist(s', X) = 1$ for any codeword s' in Y, then $s = s'$ and go to Step 2
  4) If $dist(s, X) > 1$, add s to set Y. This codeword cannot be decoded as of yet but may be decoded in future.

Each decoder will be 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 *F* is given a *F*ixed 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) Choose symbol s_i. 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 *F* is more constrained than the decoder *D* since it considers symbols in a fixed order only and throws away all symbols which cannot be decoded immediately but might have been useful at a later point in the decoding process. More formally, this can be stated as follows:

Lemma 5.2: Given a sequence of codewords $\sigma_{[1,k]} = s_1, s_2, \ldots, s_k$ of size $\|\sigma_{[1,k]}\| = k$, $D(\sigma_{[1,k]}) \geq F(\sigma_{[1,k]})$.

Proof: Recall that decoder *D* does not decode the codewords in a fixed order but rather decodes the degree 1 codewords first and from then on, decodes the codeword with the minimum distance from the set of recovered symbols.

Let $D_N(\sigma_{[1,k]})$ be the number of symbols decoder *D* can recover from the set of codewords $\sigma_{[1,k]} = s_1, \ldots, s_k$ when they are constructed from a set of N original symbols $x_1, \ldots, x_N$.

Similarly, let $F_N(\sigma_{[1,k]})$ be the number of symbols decoder *F* can recover from the *fixed sequence* of codewords $\sigma_{[1,k]} = s_1, \ldots, s_k$.

We will prove $D_N(\sigma_{[1,k]}) \geq F_N(\sigma_{[1,k]})$ by induction on the number of codewords k in $\sigma_{[1,k]}$.

- **[Basis Step: k=1]:** Clearly $D_N(s_1) \geq F_N(s_1)$ since in this case $\sigma_{[1,1]} = s_1$. Specifically, if $degree(s_1) = 1$, then $D_N(s_1) = F_N(s_1') = 1$ and $D_N(s_1) = F_N(s_1') = 0$ otherwise.
  This holds for all positive values of N.
- **[Induction Hypothesis]:** $D_N(\sigma_{[1,k-1]}) \geq F_N(\sigma_{[1,k-1]})$. Again, this holds for all positive N.
- **[Inductive Step]:** There are two cases:
  1) [$degree(s_1) > 1$]: In this case, decoder *F* cannot decode codeword s_1. Recall that decoder *F* does not use a codeword once it has been considered. Hence, $F_N(\sigma_{[1,k]}) = F_N(\sigma_{[2,k]})$. By the induction hypothesis, we have $F_N(\sigma_{[2,k]}) \leq D_N(\sigma_{[2,k]}) \leq D_N(\sigma_{[1,k]})$

2) [$degree(s_1) = 1$]: In this case, decoder *F* is able to decode s_1 and $F_N(\sigma_{[1,k]}) = 1 + F_{N-1}(\sigma_{[2,k]})$ since symbol $x_1 = s_1$ is now recovered and the rest of the codewords are effectively constructed from $N-1$ original symbols ($x_2, \ldots, x_N$, after subtracting x_1 in all codewords containing it).

Decoder *D* decodes all degree 1 symbols in the beginning. Since s_1 is of degree 1, decoder *D* is able to decode s_1. Hence, $D_N(\sigma_{[1,k]}) = 1 + D_{N-1}(\sigma_{[2,k]})$.

Again, from the induction hypothesis, since $\|\sigma_{[2,k]}\| = k-1$, we have, $F_{N-1}(\sigma_{[2,k]}) \leq D_{N-1}(\sigma_{[2,k]})$ and hence $D_N(\sigma_{[1,k]}) \geq F_N(\sigma_{[1,k]})$.

■

We define another decoder *S*, which is a special case of decoder *F*.

Definition 5.3: Decoder *S* is given a set of encoded symbols $s_1, s_2, \ldots, s_k$, works as follows: Initially the set X of recovered symbols is empty.

1) Sort the symbols in non-decreasing order of their degrees to get the sequence $s'_1, s'_2, \ldots, s'_k$.
2) Run decoder *F* on the sequence $s'_1, s'_2, \ldots, s'_k$.

Decoder *S* is essentially a special case of decoder *F* which works on the 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.

B. Finding the best-degree codeword for S *and* F

Let us consider our decoders *S* and *F* which do not maintain previously received codewords. When attempting to decode, they can only utilize the data that they have 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 5.4: Let $\rho_{r,d}$ be the probability of successfully decoding a degree d symbol when r symbols have already been recovered. Then $\rho_{r,d} = \dfrac{\binom{r}{d-1}(N-r)}{\binom{N}{d}}$.

Proof: Recall that we assume that the d symbols of the degree d symbol are assumed to be distinct and uniformly chosen without replacement from the N symbols. A new symbol can be decoded from this codeword of degree d if all but 1 of the d components of the codeword have already been recovered.

The number of ways of choosing a d degree symbol such that the component symbols are distinct and are spread uniformly randomly is $\binom{N}{d}$. There are r recovered symbols and $N-r$

unrecovered symbols. For a degree d symbol, the number of ways of choosing 1 component from the $N-r$ unrecovered symbols is $\binom{N-r}{1}$. Similarly, the number of ways of choosing $d-1$ components from the set of r recovered symbols is $\binom{r}{d-1}$.

The probability that for a degree d symbol, $d-1$ components are from the set of r recovered symbols and 1 from the set of $N-r$ unrecovered symbols is $\rho_{r,d} = \frac{\binom{r}{d-1}(N-r)}{\binom{N}{d}}$. ■

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 our toy model problem considered previously:

$$R_1 = \frac{N-1}{2}, R_2 = \frac{2N-1}{3}, \ldots, R_i = \frac{iN-1}{i+1} \forall_{i \in [1, N-1]}. \tag{2}$$

Lemma 5.5: $\rho_{r,i} \geq \rho_{r,i+1}$ as long as $r \leq R_i = \frac{iN-1}{i+1}$

Proof:

$$\begin{aligned} \rho_{r,i} &\geq \rho_{r,i+1} \\ \Rightarrow \frac{\binom{r}{i-1}(N-r)}{\binom{N}{i}} &\leq \frac{\binom{r}{i}(N-r)}{\binom{N}{i+1}} \\ \Rightarrow \frac{N-i}{i+1} &\geq \frac{r-i+1}{i} \\ \Rightarrow r &\leq \frac{iN-1}{i+1} \end{aligned} \tag{3}$$

■

The above result validates our earlier intuition that when the number of recovered symbols is low, low degree codewords are better whereas as the number of recovered symbols increases, higher and 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 5.6: If $\rho_{r,i} < \rho_{r,j}$ for some $i < j$, then $\rho_{r',i} < \rho_{r',j}$ for any $r' > r$

Proof:

$$\begin{aligned} \rho_{r,i} &< \rho_{r,j} \\ \Rightarrow \frac{(j-1)!(r-j+1)!}{(i-1)!(r-i+1)!} &< \frac{(i)!(N-i)!}{(j)!(N-j)!} \\ \Rightarrow \frac{(r-j+1)!}{(r-i+1)!} &< \frac{(i-1)!(i)!(N-i)!}{(j-1)!(j)!(N-j)!} \end{aligned} \tag{4}$$

We will prove this by contradiction. Lets assume now that $\rho_{r',i} > \rho_{r',j}$ for $r' > r$. Which means

$$\begin{aligned} \rho_{r',i} &> \rho_{r',j} \\ \Rightarrow \frac{(r'-j+1)!}{(r'-i+1)!} &> \frac{(i-1)!(i)!(N-i)!}{(j-1)!(j)!(N-j)!} \end{aligned} \tag{5}$$

Inequalities 4 and 5 imply that

$$\begin{aligned} \frac{(r'-j+1)!}{(r'-i+1)!} &> \frac{(r-j+1)!}{(r-i+1)!} \\ \Rightarrow (r'-j+1)\dots(r-j) &> (r'-i+1)\dots(r-i) \\ \Rightarrow \frac{(r'-j+1)}{(r'-i+1)}\dots\frac{(r-j)}{(r-i)} &> 1 \end{aligned} \tag{6}$$

which is clearly impossible since each term on the LHS is less than 1 since $i < j$. Hence, our assumption was false which proves the lemma. ■

The result basically states that once degree j codewords have become more useful than lower degree codewords, they will remain so even when more symbols have been 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.

C. Decoder results

Lemma 5.7: When using decoder *S*, to recover more than r symbols, the optimal degree distribution contains symbols of degree j only if $\forall_{1\leq i<j}[\rho_{r,i} < \rho_{r,j}]$.

Proof: Decoder *S* decodes the symbols in the ascending order of their degrees. According to Lemma 5.5, $\rho_{r,d}$ is maximized for $d = 1$ when $r \leq R_1$. For $R_1 < r \leq R_2$, $\rho_{r,d}$ is maximized for $d = 2$ and so on. According to Lemma 5.6, once $\rho_{r,d}$ is smaller than $\rho_{r,d+1}$, it will always be so when more symbols have been recovered.

These two observations coupled with the fact that decoder *S* does not store any symbols which cannot be immediately decoded, it follows that to recover the $r+1^{st}$ symbol, the optimal degree distribution for decoder *S* will have a degree j codeword only if $\rho_{r,j} > \rho_{r,j-1}$.

It easily follows from Lemmas 5.5 and 5.6, that $\forall_{1 \leq i < j}[\rho_{r,j} > \rho_{r,i}]$.

■

D. 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 5.8: To recover r symbols, such that $r \leq R_1 = \dfrac{N-1}{2}$, the optimal degree distribution has symbols of only degree 1 when using decoder *D*

Proof:

We will prove this by induction on r.

- [Basis Step: r=1]: Clearly, to recover the first symbol, only a single degree 1 codeword is sufficient for decoder *D*.
- [Induction Hypothesis]: To recover $r-1$ symbols ($r-1 \leq \frac{N-1}{2}$), the optimal degree distribution has only degree 1 codewords.
- [Inductive Step]: To recover the first $r-1$ symbols using decoder *D*, the optimal degree distribution has only degree 1 codewords. To recover the r_th symbol, either a degree 1 or a higher degree codeword can be used. In either case, decoder *D* behaves the same as decoder *S* since the codewords are in ascending order of their degrees.

  According to Lemma 5.7, decoder *S* uses only degree 1 symbols to recover the first r symbols as long as $\rho_{r,1} > \rho_{r,2}$. This is true for $r \leq r_1 = \frac{N-1}{2}$.

  The result follows.

■

Theorem 5.9: To recover $R_1 = \dfrac{N-1}{2}$ data units, the expected number of encoded symbols required is $K_1 = \displaystyle\sum_{i=0}^{R_1-1} \frac{N}{N-i}$ when using decoder *D*.

Proof: From Theorem 5.8, the optimal degree distribution will contain only degree 1 symbols to recover the first $R_1 = \frac{N-1}{2}$ symbols. Since all the encoded symbols have to be

degree 1, the optimal code can be analyzed via the well studied *Coupon Collector's Problem*. To get the first r distinct coupons, one needs to collect $\sum_{i=0}^{r-1} \frac{N}{N-i}$ coupons in expectation.

Hence the expected number of symbols required is $K_1 = \sum_{i=0}^{R_1-1} \frac{N}{N-i}$.

■

If $s_1, s_2, \ldots, s_k$ is a set of k encoded symbols, then $D(s_1, \ldots, s_k)$ is the number of symbols that can be recovered from these symbols using decoder *D*.

Theorems 5.8 and 5.9 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 5.11 and 5.12. But before that we prove a lemma which is useful for the theorems.

Lemma 5.10: If X is a set of r symbols recovered by decoder *D* at any point during its execution and $a_1, \ldots, a_k$ are the remaining symbols to be considered then $E[D(X, a_1, \ldots, a_k)] \geq E[D(X', a_1, \ldots, a_k)]$ if $\|X\| \geq \|X'\|$ where X' is another set of recovered symbols

Proof: We will prove this by induction on k.

- [Basis Step: k = 0]: Since $\|X\| > \|X'\|$, trivially $E[D(X)] \geq E[D(X')]$.
- [Induction Hypothesis]: $E[D(X, a_1, \ldots, a_{k-1})] \geq E[D(X', a_1, \ldots, a_{k-1})]$ for $\|X\| > \|X'\|$
- [Inductive Step:] Let $X_{k-1} = r$ be the set of recovered symbols after decoding $a_1, \ldots, a_{k-1}$ starting with X. Similarly, let $X'_{k-1} = r'$ be the set of recovered symbols after decoding $a_1, \ldots, a_{k-1}$ starting with X'. According to the induction hypothesis, $\|X_{k-1}\| > \|X'_{k-1}\|$ which means $r \geq 1 + r'$ since both r and r' are integral.

  $D(X, a_1, \ldots, a_k) = \|X_{k-1}\| + p_{a_k}$ where p_{a_k} is the probability that a_k has distance 1 from set X_{k-1}. Similarly, $D(X', a_1, \ldots, a_k) = \|X'_{k-1}\| + p'_{a_k}$ where p'_{a_k} is the probability that a_k has distance 1 from set X'_{k-1}.

  Hence,

$$\begin{aligned} D(X, a_1, \ldots, a_k) &- D(X', a_1, \ldots, a_k) \\ &= r + p_{a_k} - r' - p'_{a_k} \\ &\geq 1 + p_{a_k} - p'_{a_k} \\ &\geq 0 \end{aligned}$$

■

Theorem 5.11: To recover r symbols such that $r \leq R_j = \dfrac{jN-1}{j+1}$, the optimal degree distribution has symbols of degree no larger than j

Proof: We will prove this by contradiction. Suppose there is an optimal degree distribution $\bar{\pi}^{opt}$ to recover r symbols such that $\sum_{i=1}^{j} \pi_i^{opt} < 1$. Consider a set of k encoded symbols $s_1, s_2, \ldots, s_k$ according to this degree distribution such that there are 1 or more symbols of degrees greater than j. WLOG, lets assume $s_1, s_2, \ldots, s_k$ is the order in which decoder *D* considers the symbols while decoding.

The expected number of symbols recovered using decoder *D* is given by $D(s_1, s_2, \ldots, s_k) \leq R_j$ according to the problem statement. Since there are symbols of degrees greater than j, let s_l be the first symbol in this sequence of degree greater than j, that is degree$(s_l) = d > j$.

Let X be the set of symbols recovered after decoding symbols $s_1, \ldots, s_{l-1}$. Let $r = \|X\|$. The expected output of the first l symbols is given by $D(s_1, s_2, \ldots, s_l) = r + p_{s_l}$ where p_{s_l} is the probability that symbol s_l of degree $d > j$ has a distance 1 from the set X. Again, according to the problem statement $r \leq R_j$.

If we replace symbol s_l of degree $d > j$ with symbol s'_l of degree j and decode the symbols using the decoder *F* then the expected output of decoder *F* given this fixed sequence of symbols $s_1, s_2, \ldots, s_{l-1}, s'_l$ is given by $F(s_1, s_2, \ldots, s_{l-1}, s'_l) = r + p_{s'_l}$.

Since $r \leq R_j = \frac{jN-1}{j+1}$, $p_{s_l} = \frac{\binom{r}{d-1}(N-r)}{\binom{N}{d}}$ and $p_{s'_l} = \frac{\binom{r}{j-1}(N-r)}{\binom{N}{j}}$, it is easy to see that $p_{s'_l} > p_{s_l}$ because $d > j$ and $r \leq R_j$.

Hence, $F(s_1, s_2, \ldots, s_{l-1}, s'_l) > D(s_1, s_2, \ldots, s_l)$. Using Lemma 5.2, we get $D(s_1, s_2, \ldots, s_{l-1}, s'_l) > D(s_1, s_2, \ldots, s_l)$. Hence the output of decoder *D* increases by replacing a symbol of degree greater than j by a symbol of degree j.

Now, using this result and Lemma 5.10 we get $D(s_1, \ldots, s_{l-1}, s'_l, s_{l+1}, \ldots, s_k)$ is at least as much as $D(s_1, s_2, \ldots, s_l, s_{l+1}, \ldots, s_k)$.

Hence by replacing higher degree symbols with symbols of degree j, the output of decoder *D* increases. Therefore, the optimal degree distribution to recover $r \leq R_j$ symbols has only symbols of degree j or less.

■

Let

$$A_j = \sum_{i=R_{j-1}}^{R_j - 1} \frac{\binom{N}{j}}{\binom{i}{j-1}(N-i)} \tag{7}$$

and

$$K_j = \sum_{i=1}^{j} A_i \tag{8}$$

Theorem 5.12: To recover $R_j = \frac{jN-1}{j+1}$ symbols, the expected number of encoded symbols required is at most K_j

Proof: We first prove that when using decoder *S*, to recover $R_j = \frac{jN-1}{j+1}$ symbols, the expected number of encoded symbols required is exactly K_j.

We will prove this by induction on j.

- [Basis Step: j = 1]: Using Lemma 5.7, to recover $R_1 = \frac{N-1}{2}$ symbols, decoder *S* uses only degree 1 codewords and hence requires an expected number of $K_1 = A_1 = \sum_{i=0}^{\frac{N}{2}-1} \frac{N}{N-i}$ codeword symbols.
- [Induction Hypothesis]: To recover $R_{j-1} = \frac{(j-1)N-1}{j}$ data units, decoder *S* requires an expected number of K_{j-1} codeword symbols.
- [Inductive Step:] Let X of size $\|X\| = R_{j-1}$ be the set of symbols recovered at some point during the execution of decoder *S*. According to Lemma 5.7, in the optimal case, decoder *S* now decodes symbols of degree j until R_j symbols have been recovered.

  When r symbols, such that $R_{j-1} \leq r < R_j$ have been recovered, the next degree j codeword will have distance 1 from the set of recovered symbols with probability $p = \frac{\binom{r}{j-1}(N-r)}{\binom{N}{j}}$.

  If in the worst case, we do not use the symbol later if it turns out to be of distance 2 or more, then the expected number of symbols required to recover the next symbol is $1.p + 2.p(1-p) + 3.p(1-p)^2 \ldots = \frac{1}{p}$.

  Hence, the expected number of codewords required by decoder *S* to recover $R_j - R_{j-1}$ symbols using degree j codewords is given by $A_j = \sum_{i=R_{j-1}}^{R_j-1} \frac{\binom{r}{j-1}(N-r)}{\binom{N}{j}}$.

  According to Lemma 5.2, given any set of codewords, decoder *D* always recovers at least as many symbols as decoder *S*. Hence, the expected number of codewords required by decoder *D* to recover R_j symbols is bounded by $K_j = \sum_{i=1}^{j} A_i$.

■

E. A New Degree Distribution based on Growth Codes

According to the analysis in section V-D, 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 some of k symbols are remaining. A degree distribution can thus be defined as

$$\bar{\pi}^*(k) : \pi_i^* = max(0, min(\frac{K_i - K_{i-1}}{k}, \frac{k - K_{i-1}}{k})) \tag{9}$$

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 $\bar{\pi}^*(k)$ with k varying 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 .

VI. At-Sink Simulation Model

In this section, we evaluate how the Growth Codes degree distribution compares to other degree distributions that are known to perform well in a *perfect source* setting, i.e. when the codewords received at the destination exactly fit a desired degree distribution.

We have the following scenario: There is 1 source and 1 sink. The source has all available symbols $x_1, \ldots, x_N$. It constructs codewords according to some degree distribution π using the encoder described in Section IV-A, i. e., a codeword of degree d is constructed with probability π_d and every codeword of degree d occurs with uniform probability. The sink receives the

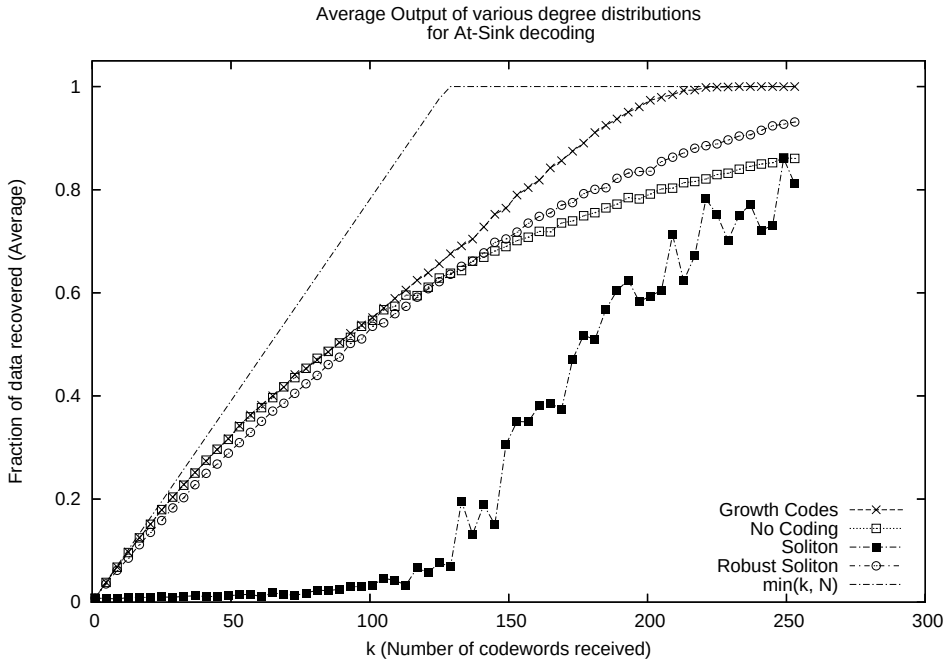


Fig. 3. Comparing the performance of various degree distributions in an At-Sink setting. N is chosen to be 128

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.

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 Table I to recover data on the fly.

We call this the At-Sink simulation model since we generate codewords according to some degree distribution and assume that all these codewords are received at the sink. This allows us to compare the Growth Codes degree distribution with other degree distributions such as *Soliton* and *Robust Soliton* for whom it is not clear how to construct codewords in a fully distributed environment to fit these desired degree distributions. Furthermore, these distributions were not specifically designed to recover data when only a small number of codewords are obtained. For the simulation, 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 [3].

Figure 3 depicts the fraction of original data recovered at the sink varying with the number of codewords received. On the X-axis, we plot the number of codewords received at the sink

and on the Y-axis, we measure what fraction of the original data can be recovered from the codewords received until that point. The results are from an average of 100 simulation runs. It can be observed that if codewords are constructed according to the Growth Codes degree distribution, the sink is able to recover more symbols at *any* given time than when using any of the other degree distributions. For a small number of received codewords, unencoded (degree 1) codewords recover the most number of symbols as can be clearly observed.

Soliton distribution does not perform very well for low number of received codewords since it generates too many high degree codewords that are useless until a substantial number of symbols have been decoded.

VII. Growth Codes in a Distributed Setting

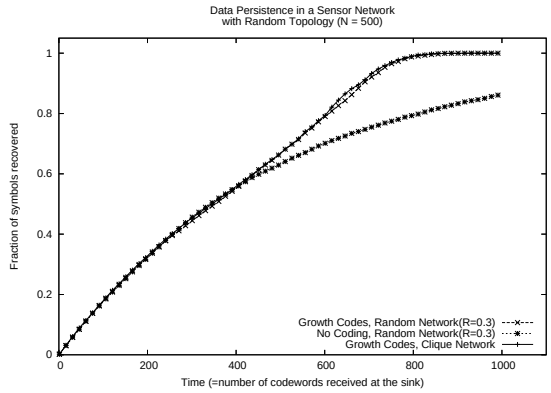
In the previous section, we saw that in a *perfect source* setting, Growth Codes recover the most data from even a small number of received codeword symbols. Degree distributions such as Soliton and Robust-Soliton are designed for recovering "all" data with the minimum number of codewords and not for recovering data from a small number of codeword symbols, although Robust-Soliton does a reasonably good job of it.

In a completely distributed setting, where each sensor node n_i generates a unit of data x_i, it is not clear how to construct codewords to fit a desired degree distribution since the nodes have limited storage and it is difficult to aggregate at a single node all the symbols required to construct a codeword.

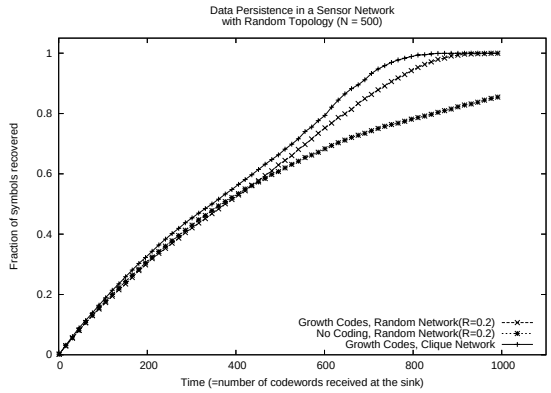
On the other hand, codes with monotonically increasing codeword complexity such as Growth Codes are easily implementable using the protocol described in Table I. Following this protocol, the nodes in the network are able to construct codewords that fit degree distributions of the kind as in Equation 9.

Hence, 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.

In the following sections, we look at how fast Growth Codes can help the sink recover data in a completely distributed environment. We evaluate them in a network of nodes connected in a random topology where nodes can communicate only when they lie within a certain euclidian distance from one another, as is assumed in modeling wireless communications [27]. Later on,



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



(b) Data recovered at the sink in a 500 node random network of radius $R = 0.2$ as more and more codewords are received

Fig. 4. Data Recovery using Growth Codes in random networks of various densities

we evaluate how Growth Codes can recover a substantial fraction of data in a disaster-prone network where parts of the network may fail at any time.

In all simulations, 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 $10 * N$ 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 did simulations with higher C values and did not find any significant difference.

A. Growth Codes in a Random Topology

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.

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.

We assume each node in the network generates a symbol x_i in the beginning. 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 Table I.

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 randomly setup with a certain radius R. The sink is attached to a random node in the network.

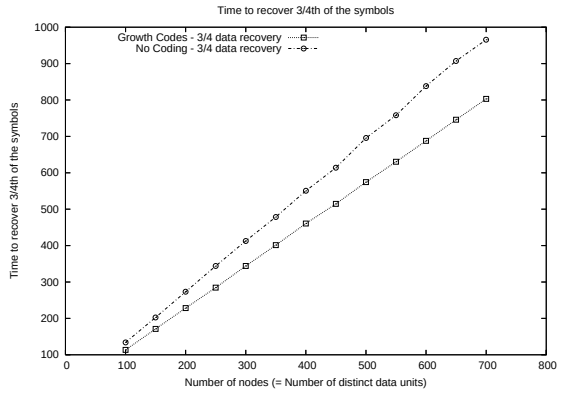
Figure 4(a) depicts the fraction of data recovered by the sink as a function of the number of received codewords in a random network with radius of $R = 0.3$. The results are an average of 100 simulation runs. On the X-axis, the number of codewords received by the sink are plotted while the Y-axis is 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.

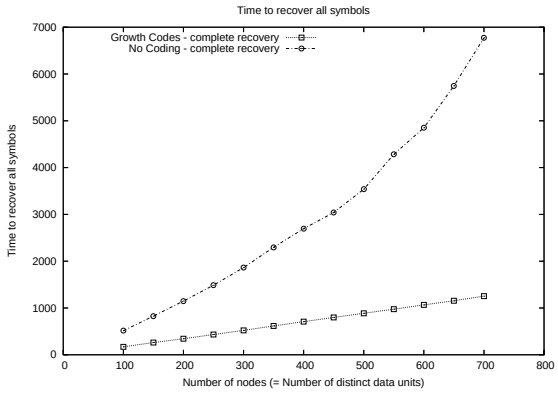
We now evaluate the data recovery rate using Growth Codes when the network has a radius of $R = 0.2$ implying that the network is much sparser. From Figure 4(b) we can observe that the data delivery rate of Growth Codes does not deteriorate very much even in a much sparser random network.

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 5(a) depicts how many rounds on average are required before the sink is able to recover $3/4^{th}$ of the original symbols. On the X-axis, we vary the number of nodes in the network which is the same as the total number of symbols (since each node generates one symbol). On the Y-axis, we measure the time taken to recover $3/4^{th}$ of the symbols. Figure 5(b) depicts the time



(a) Time taken to recover $3/4^{th}$ of the data in a 500 node network when the nodes use Growth Codes versus no coding



(b) Time taken to recover all the data in a 500 node network when the nodes use Growth Codes versus no coding

Fig. 5. Recovery rate using Growth Codes

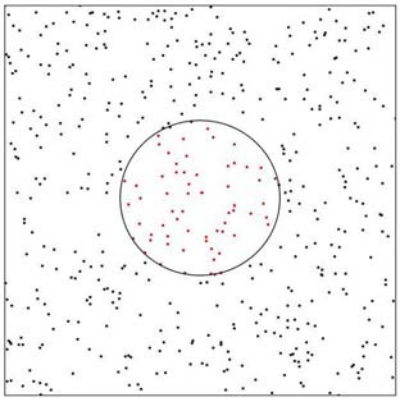
taken to recover *all* the symbols.

To recover $3/4^{th}$ of the data, both protocols require codewords 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. It takes a long time to acquire the last few symbols when not using any coding. The total time is $O(NlogN)$ as studied in the *Coupon Collector's Problem*.

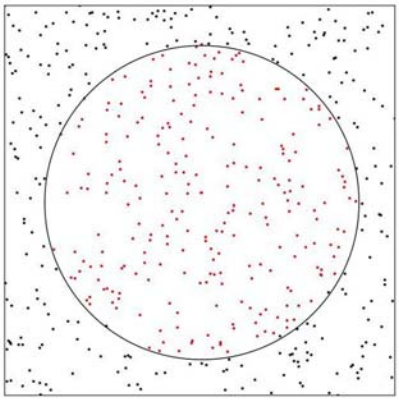
B. Growth Codes in Disaster-Prone Networks

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 a parameter r which is the radius of the impact. At the time of the disaster, all nodes within r distance of the center of the impact are disabled. This in effect, changes



(a) Portion of the network disabled when impact radius $r = 0.2$



(b) Portion of the network disabled when impact radius $r = 0.4$

Fig. 6. Fraction of network disabled in case of disasters of different impact radii. The nodes inside the circle (red nodes) are disabled at the time of impact. The sink is attached to the top-leftmost node

the topology of the network since the disabled nodes and all links connecting them become non-functional.

Figure 6(a) depicts the portion of the network that fails at the time of the disaster if the radius of impact is $r = 0.2$. The center of impact is located at the center of the square region. We attach the sink to the top-leftmost sensor node in the network. Figure 6(b) displays the portion of the network that fails at the time of the disaster if the radius of impact is $r = 0.4$.

We discuss how the time of disaster and the radius of impact affects the network's ability to deliver data to the sink. Figure 7 shows how much data can be recovered at the sink with time 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. 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

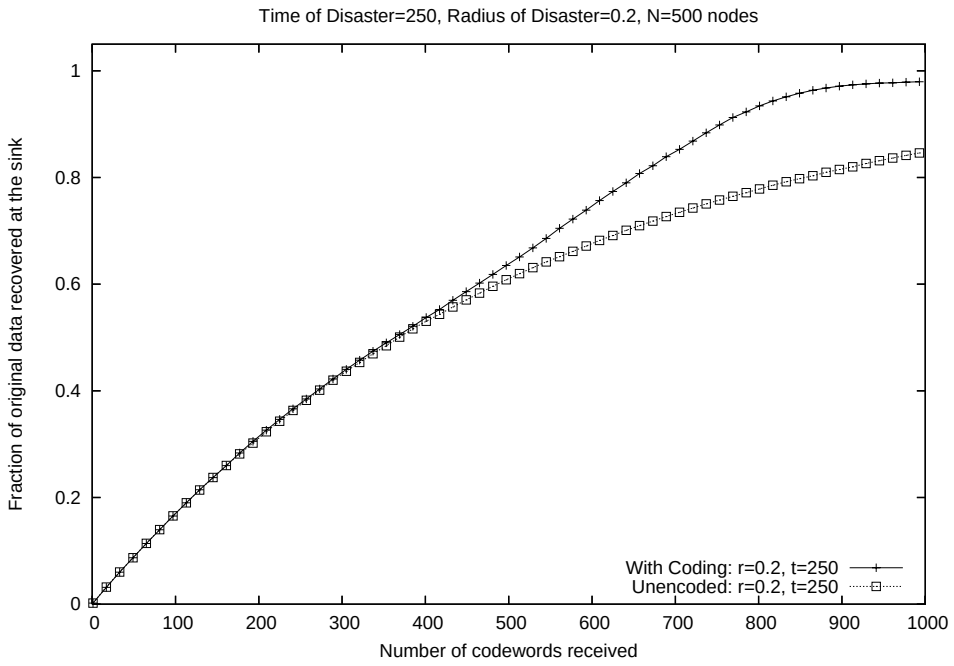


Fig. 7. Data recovered at the sink in a 500 node random network in a disaster scenario of impact radius $r = 0.2$ at time $t = 250$

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 8, 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 proportional to the square of the impact radius r. On the Y-axis, we display how long does it take for the sink to recover as much data as has survived the disaster. We also display how many of the original 500 symbols survive in each of the cases. 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.

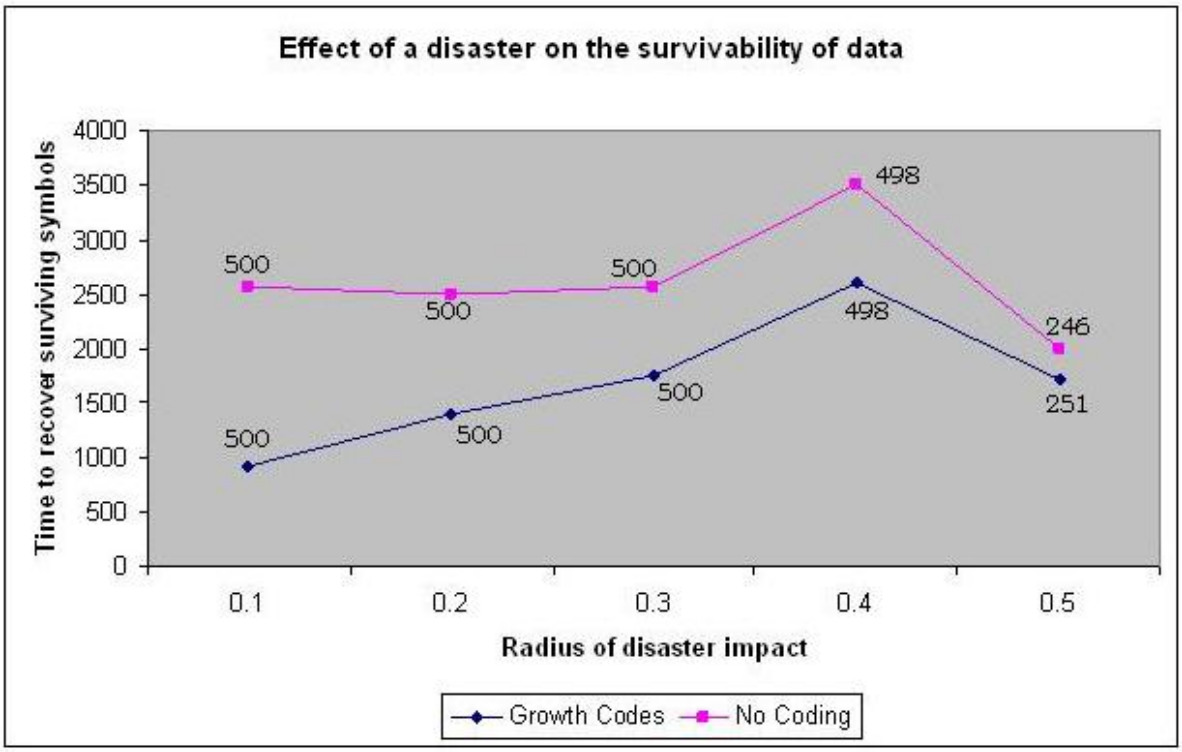


Fig. 8. Time required to recover surviving data for a disaster at time $t = 50$ of varying intensities. The number inside the figure are the number of symbols out of the original 500, that can ultimately be recovered at the sink

VIII. Growth Codes in Practice

In Sections VI and VII, we observed that Growth Codes perform well in various network scenarios and provide a method for data to persist in sensor networks and be rapidly propagated to the sink even when portions of the network fail. We discuss some practical issues concerning the implementation of such codes in real networks.

As discussed in Section III-B, sensor nodes generate data periodically. We call each such time period an *epoch*. Our Growth Codes protocol, divides each epoch into discrete rounds. This is done merely to facilitate the description and analysis of our techniques. All that is necessary for the protocol to work is that there be a globally consistent timestamp for each epoch. Otherwise, nodes can exchange data asynchronously and the assumption of discretization of the time line can be easily relaxed.

In our simulations, we did not touch upon the case when the sensor network is alive for multiple epochs and nodes generate data at every epoch. In case the data generation at the nodes

is much faster than the rate at which the sink recovers symbols, a design issue that comes up is how and when to phase out older data from the network. One solution is for the nodes to flush out all codewords from their storage and fill them up with their newly generated data at every epoch. Irrespective of the data generation rate, using Growth Codes the sink will be able to decode a substantial number of the received codewords from each epoch.

As discussed in Section V, we use $K_i's$ (which are upper bounds) as transition points to higher degree codewords. The optimal transition facilitate recovery of symbols from a minimum expected number of received codewords. Since the Growth Codes protocol has in-built randomness at many stages, the codewords constructed may not fit exactly the desired degree distribution. Taking a conservative approach and using higher than optimal transition points prevents the sink from accumulating too many useless high degree symbols.

Another design problem is the size of high degree codewords. For a degree d codeword, one needs to specify which d of the N symbols are XOR-ed together. As was suggested by Chou et. al. [28], the coefficient size is negligible if the data symbol is large enough. Even when the data symbol is not large enough, coefficient size can be decreased by limiting the maximum degree of codewords constructed. This is again a bit conservative but will help take care of the variance of the codewords constructed in the network.

IX. 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 or earthquakes where parts of the network may get destroyed at any time.

We also design a protocol based on Growth Codes which can be used by sensor network nodes to encode and distribute data. This protocol does not require any knowledge of the general network topology or of the sink locations. Therefore, Growth Codes are well suited for deployment in *zero-configuration* networks: networks where data collection and transmission

must be initiated immediately, before nodes have the opportunity to learn about the specifics of the network, aside from some limited information about their immediate surroundings.

We propose a dynamically changing codeword degree distribution which suggests construction of low degree codewords at the start and transitioning to higher degrees later on. We provide upper bounds for the optimal transition points which result in maximum data retrieval from a given number of codewords.

We compare, via simulations, the Growth Codes protocol with other coding schemes in various network scenarios and show that it performs better than any other coding scheme in typical network settings.

REFERENCES

[1] S. S. Pradhan, J. Kusuma and K. Ramchandran, "Distributed compression in a dense microsensor network," in *IEEE Signal Processing Magazine*, vol. 19, no. 2, Mar. 2002, pp. 51–60.

[2] 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.

[3] M. Luby, "LT Codes," in *Symposium on Foundations of Computer Science*, 2002.

[4] 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.

[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] R. Ahlswede, N. Cai, S. Y. R. Li and R. W. Yeung, "Network Information Flow," in *IEEE Transactions on Information Theory*, vol. 46, 2000, pp. 1004–1016.

[7] N. Cai, S. Y. R. Li and R. W. Yeung, "Linear Network Coding," in *IEEE Transactions on Information Theory*, vol. 49, no. 2, 2003, pp. 371–381.

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

[9] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance," in *Journal of the ACM*, vol. 36, no. 2, Apr. 1989, pp. 335–348.

[10] R. Koetter and M. Medard, "An Algebraic Approach to Network Coding," in *ACM/IEEE Transactions on Networking*, vol. 11, no. 5, 2003, pp. 782–795.

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

[12] A. Albanese, J. Blömer, J. Edmonds, M. Luby, M. Sudan, "Priority Encoding Transmission," in *IEEE Transactions on Information Theory*, vol. 42, no. 6, Nov. 1996.

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

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

[15] M. Luby, M. Mitzenmacher, M. A. Shokrollahi and D. Spielman, “Efficient Erasure Correcting Codes,” in *IEEE Transactions on Information Theory*, vol. 47, no. 2, 2001, pp. 569–584.

[16] 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.

[17] 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*, 2003, pp. 116–123.

[18] 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.

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

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

[21] S. Palchaudhuri, A. K. Saha and D. B. Johnson, “Adaptive Clock Synchronization in Sensor Networks,” in *Information Processing in Sensor Networks*, 2004, pp. 340–348.

[22] Q. Li and D. Rus, “Global Clock Synchronization in Sensor Networks,” in *Proceedings of INFOCOM*, 2004.

[23] 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*, 2004, pp. 28–35.

[24] 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*, Mar. 2004, pp. 71–80.

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

[26] T. Ho, M. Mdard, M. Effros and D. Karger, “On Randomized Network Coding,” in *Allerton Annual Conference on Communication, Control and Computing*, Oct. 2003.

[27] 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.

[28] P. A. Chou, Y. Wu, and K. Jain, “Practical Network Coding,” in *Allerton Annual Conference on Communication, Control and Computing*, 2003.