Network Coding Strategies for Data Persistence in Static and Mobile Sensor Networks

Daniele Munaretto†, Jörg Widmer⋆, Michele Rossi†, and Michele Zorzi†
†DEI, University of Padova, via Gradenigo 6/B – 35131 Padova, Italy
⋆DoCoMo Euro-Labs, Landsberger Strasse 312 – 80687 Munich, Germany

***Abstract*—Network coding promises to bring significant performance improvements to sensor network protocols but algorithms need to be designed to cope with the often very constrained resources of sensor nodes. Growth Codes proposed by Kamra et al. [1] are one such example aimed at improving sensor network data persistence. The codes use simple coding operations and require comparatively little memory. However, Growth Codes are based on the assumption of an extremely dynamic network topology and do not perform well in more stable settings. In this paper we propose modifications to Growth Codes that are able to achieve good performance over a wider range of static and dynamic scenarios. In particular, we investigate changes of how many and which symbols the transmitted information is coded over and how the decoding is performed. These modifications are analyzed in detail by means of simulations.**

I. Introduction

Sensor networks are special purpose networks usually concerned with data collection and efficient transport of that data to one or several sink nodes. The most important characteristics of sensor networks are the constraints imposed by the limited resources of sensor nodes in terms of memory, computational power, and energy. Network coding [2] has proven to be a useful tool to improve the performance of such networks. Several papers have analyzed the benefits of random network coding [3] for information dissemination and data persistence [4]. In particular, the robustness that can be achieved through the diversity of available information by coding at intermediate nodes can be quite important in the fragile environment of sensor networks, where node failures can be common.

For practical algorithms, it is important that they are designed with consideration of the aforementioned resource constraints. In this paper we focus on network coding based algorithms for data dissemination in sensor networks. We are particularly interested in increasing data persistence in the face of network failures, where *data persistence* is defined as the fraction of data generated within the network that eventually reaches the sink. Recently, algorithms such as Growth Codes [1] were introduced which are particularly suitable for these scenarios. The design of Growth Codes is based on the assumption that the network topology changes very rapidly, e.g., due to link/node failures or mobility. In this case, it makes sense for nodes to initially mainly disseminate their own data and then to gradually code over other data as well, to increase the probability that information survives. However, in static scenarios this behavior is detrimental to performance. In case the set of neighbors does not change significantly from one transmission to the next, the repeated transmission of the same data will not bring new information to the neighbors and in turn will prevent the propagation of other information. Realistic scenarios are likely to be in between these two extremes and have some coherence in the composition of the neighborhood over time.

In such cases, it is beneficial to forward and code over information from other nodes with higher probability and to increase the codeword degree (i.e., the number of symbols a codeword is coded over) more aggressively than in [1]. We investigate which degree distributions improve the probability of early decoding, and thus of data recovery, in case of node failures. We further propose algorithms that require only a very limited increase of coding complexity and memory usage. Our claims are supported by extensive simulation results. The paper is structured as follows. In Section II we give an overview of related work. In Section III we look at existing algorithms based on network coding from the point of view of codeword degree distributions, and present some modifications to Growth Codes that can improve performance in more general settings. Section IV provides detailed simulation results for the different algorithms, and Section V concludes the paper.

II. Related Work

The usefulness of network coding for data storage was investigated in [4], where the authors showed that a simple distribution scheme using network coding and based only on local information can perform almost as well as the case where there is complete coordination among nodes. Similar considerations also apply to sensor networks. The authors of [5], [6] investigate algorithms for the extreme case where storage nodes can only store a single (coded or uncoded) information unit. Data is pre-routed to $\log(N)$ of the N storage nodes and the storage nodes simply combine the incoming information with their existing information (i.e., the received codeword is multiplied with a random coefficient and added to the existing codeword). The algorithm ensures that nodes can decode the k information units of interest by querying a corresponding number of storage nodes. In [7], the authors extend this approach by showing that if these conditions are slightly relaxed, a constant pre-routing degree suffices.

The authors of [8] present a simple network coding algorithm that achieves optimum performance for broadcast data dissemination in static and symmetric grid networks. Optimality here means that *each* broadcast transmission of a node brings one

codeword worth of new information to *each* neighbor. The algorithm progresses iteratively, where the codewords transmitted during a given iteration are coded over the codewords received during the previous iteration. The coding process ensures that nodes can successively decode information generated at neighbors at further distance, i.e., first all 1-hop neighbors, then all 2-hop neighbors, etc. This mechanism provides a substantial improvement in overhead compared to flooding, where forwarded packets have usually already been received by a number of neighbors previously, and thus increase overhead without bringing new information to those neighbors. The mechanism is quite simple, but not specifically tuned to the resource constraints of sensor nodes. In particular, the number of symbols a codeword is coded over increases with the size of the neighborhood from which the symbols originate.

In contrast, Growth Codes [1] were specifically designed with these limitations in mind. Sensor nodes send out codewords that can be coded over multiple original information units. Nodes exchange codewords with their neighbors and combine received codewords with the existing local information, such that, the stored information is coded over more and more information units over time. Storage space is limited and newly received codewords will overwrite older ones. To recover information, a data collector node has to retrieve the stored information units of a sufficient number of sensor nodes and then decode. The number of original information units a stored codeword is coded over is called codeword degree. The authors propose to gradually increase the codeword degree with the amount of received information, hence the name "Growth Codes". This codeword degree distribution optimizes sensor network data persistence under node failure, as it allows to decode the joint information of any subset of nodes with high probability. Intuitively, a high degree increases the probability that transmissions are innovative in that they bring new information to the neighbors, while a low degree increases the probability that the information can be decoded immediately upon reception, thus decreasing the likelihood that nodes will be left with undecodable information in case parts of the sensor network fail.

A scheme that uses a codeword degree distribution that is between those of the broadcasting algorithm and of Growth Codes is analyzed in [9]. The nodes send out codewords that have a degree uniformly distributed between one and the maximum possible degree (i.e., coding over all the available information). The authors argue that the algorithm facilitates the removal of obsolete information units from the sensor network. Its performance is investigated by means of simulation for static uniformly random network topologies. However, the algorithm is designed for a very specific setting where all nodes observe the same phenomenon (i.e., they produce the same data), no communication between nodes takes place, and all transmissions are single hop directly to the data collection node.

III. Coding Algorithms

In this section, we give an overview of network coding algorithms for broadcast, as well as Growth Codes. Specifically, we look at codeword degree (i.e., the number of original symbols a transmitted codeword is coded over) required to ensure that transmissions 1) bring new information to the neighbors that receive it, and 2) can be decoded by these neighbors immediately with high probability. We then present some modifications of such algorithms that are able to improve performance in certain conditions.

A. Optimum Coding for Symmetric Topologies

We first give more details of the network coding algorithm for broadcasting presented in [8]. Assume a symmetric network of N nodes where each node i has a single symbol x_i to transmit to all other nodes. Let $\mathcal{N}_i(d)$ be the set of nodes that are at d-hop distance from node i. For the sake of simplicity, we first discuss the algorithm for $|\mathcal{N}_i(1)| = 4$, i.e., each node has exactly four direct neighbors and nodes are placed on a grid as shown in Fig. 1(a). Thus, $|\mathcal{N}_i(d)| = 4d$, $\forall i$ and $d \geq 1$ in an infinite grid.

For the coding, random linear network coding over a finite field $\mathbb{F}_q$ is used, as presented for example in [10]. The algorithm progresses in iterations, during which nodes first transmit codewords to their neighbors and in turn receive codewords from their neighbors. More specifically, in each iteration k, each node sends out a set of codewords $Y_i(k)$. Let $S_i(k)$ be the information subspace spanned by the codewords received by node i in iteration k,

$$\begin{aligned} S_i(k) &= \text{span}\left(Y_j(k) \,|\, j \in \mathcal{N}_i(1)\right) \text{ for } k \geq 1 \\ S_i(0) &= \text{span}\left(x_i\right). \end{aligned}$$

Initially, each node will once broadcast its own symbol $Y_i(1) = \{x_i\}$ uncoded to all neighbors. Because of symmetry, each node receives four symbols and can decode the information contained in its direct neighborhood.

In the next iteration $k = 2$, each node i transmits 2 random vectors from $S_i(1)$, i.e., coded over the one-hop information (4 symbols) received previously. In addition, we have that

$$\begin{aligned} \dim(S_i(1)) &= 4 \\ \dim(S_j(1) \cap S_l(1)) &\leq 2 \quad \forall\, j, l \in \mathcal{N}_i(1), j \neq l. \end{aligned}$$

Each node will thus receive 8 linearly independent codewords coded over the original information of nodes that are at most at a two-hop distance. Since all the one-hop information as well as the original packet x_i is already available at node i, the 8 received codewords (together with the already available information) allow each node i to decode all x_j, $j \in \mathcal{N}_i(2)$, as discussed in greater detail in [8]. In general, during iteration $k > 1$, a node i will send k codewords coded over the $4(k-1)$ symbols contained in its $(k-1)$-hop neighborhood (i.e., $|Y_i(k)| = k$). It will in turn receive $4k$ codewords from its direct neighbors, which allows it to decode the information of the $4k$ nodes in its k-hop neighborhood.

Since in the generic iteration j a node transmits j packets, it will take $\sum_{j=1}^{k-1} j = \frac{k(k-1)}{2}$ transmissions per node to reach

the end of iteration $k-1$. Due to symmetry, each node will at the same time have (received and) decoded[1]

$$r = 2k(k-1)$$

codewords. Hence, r is the total number of codewords decoded by a node up to and including iteration $k-1$. In addition, codewords are always coded over *all* symbols decoded in the previous iteration, and thus the degree of the codewords transmitted in iteration k corresponds to the number of original information vectors contained in the $(k-1)$-hop neighborhood from which the codewords are created,

$$\deg(y) = 4(k-1) \quad \forall\, y \in Y_i(k).$$

The degree at step k (previous equation) can equivalently be written as a function of r as follows:

$$\deg(y) = 2(\sqrt{2r+1} - 1) \quad \forall\, y \in Y_i(k).$$

Thus, the degree of the codewords grows with the square root of the number of decoded packets. This allows nodes to decode information from other nodes at successively increasing distance. Individual codewords cannot be decoded immediately, but only after all other codewords for the same k-hop neighborhood have been received.

Similar considerations also hold for regular topologies with $|\mathcal{N}_i(1)| = \Delta$ for $\Delta > 2$ (under the constraint that the Δ allows a symmetric topology). In particular, we consider the case where the topology is a regular tiling of polygons (square, hexagon), or where an increased transmission range allows nodes to reach a higher number of neighbors with a direct transmission, e.g., the 1-hop *and* 2-hop neighbors of the node shown in Fig. 1(a). In the general case, $r = \Delta\frac{k(k-1)}{2}$ and the codeword degree scales with

$$\deg(y) = \Delta(k-1) = \frac{1}{2}(\sqrt{\Delta^2 + 8r\Delta} - \Delta) \quad \forall\, y \in Y_i(k).$$

A possible example is given in Fig. 1(b), where the grid is hexagonal. In this case the number of neighbors for each node is again constant and equal to $\Delta = 6$. Moreover, as can be easily checked by inspecting the figure, we have that the k-th neighborhood of a given tagged node contains exactly $|\mathcal{N}_k| = |\mathcal{N}_{k-1}| + \Delta = k\Delta$ nodes. In the special case of a circular network or infinite line network, the number of neighbors does not increase with distance and the codeword degree is thus constant.

B. Growth Codes

For the sake of simplicity, let us again assume that each of the N nodes in the network has a single symbol (codeword) to transmit. The available storage space at a node is limited. With Growth Codes (GC) [1], each node initializes the whole storage with its own symbol. Whenever a new codeword is received, the node stores it at a random position in memory, overwriting the previously stored information. To transmit, a node randomly picks a codeword from memory and sends it out.

[1]Each node, up to and including iteration $k-1$, will receive $\frac{k(k-1)}{2}$ packets from each of its 4 neighbors.

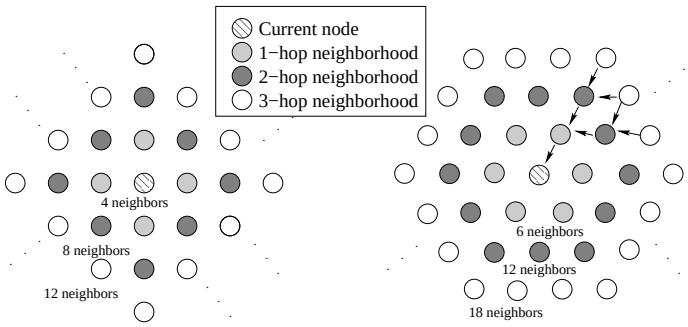


Fig. 1. Nodes on (a) a square grid and on (b) a hexagonal grid.

Since GC initializes the memory with the node's own symbol and randomly picks codewords to be transmitted from memory, the probability of transmitting own information is initially very high and then gradually decreases as the memory is filled with other codewords.

When a node receives a codeword that does not already include the node's own symbol, it may decide to `xor` the codeword with its own symbol and store the combined information. This increases the degree of the codeword by one. On the one hand, it also increases the probability that when this codeword is transmitted it provides innovative information. On the other hand, more codewords and more operations are necessary to decode it (i.e., it is more likely that the codeword cannot be decoded). If the codeword cannot be decoded immediately, it is stored and the node attempts to decode it later when more symbols are available. Whether a node is permitted to increase the degree is determined by a degree distribution which ensures, that nodes can immediately decode the received codewords with high probability.

In [1], the following degree distribution is proposed. For the first $R_1 = \left\lceil \frac{N-1}{2} \right\rceil$ symbols, codewords of degree 1 (i.e., uncoded symbols) are most likely to allow successful decoding. In general, after decoding of $R_k = \left\lceil \frac{kN-1}{k+1} \right\rceil$ symbols (for $k = 1, ..., N$), codewords of degree $k+1$ are most useful for decoding. Thus, a node is allowed to send out codewords of degree k when it has decoded between R_{k-1} and R_k symbols. The degree of a codeword y to be transmitted after r symbols have been decoded is given as

$$\deg(y) = \left\lfloor \frac{N+1}{N-r} \right\rfloor.$$

The corresponding analysis is based on the assumption that all symbols have the *same* probability of being contained in a codeword of a certain degree. Clearly, this assumption only holds if nodes encounter other nodes with uniform probability, i.e., the neighborhood of a node when transmitting a codeword is uncorrelated with the neighborhood during previous transmissions. In less random scenarios, coding performance will be sub-optimal.

C. Discussion on the Degree Distributions

In Section III-A and III-B we discussed degree distributions for random linear coding and Growth Codes, respectively. The former strategy is suitable for static networks, whereas the latter

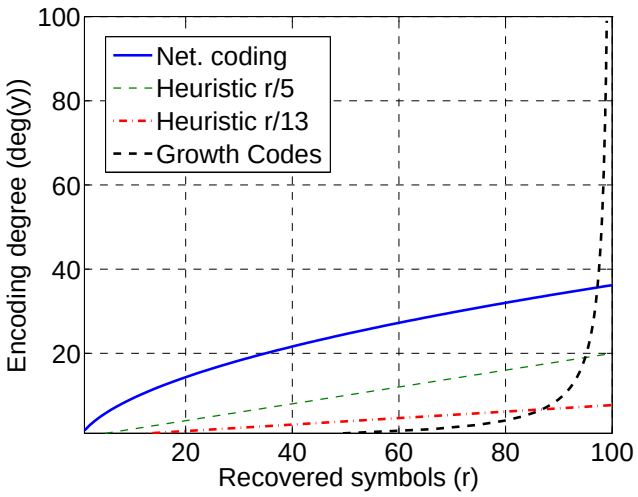


Fig. 2. Codeword degree vs. number of decoded symbols for a network of size $N = 100$.

is designed to be effective in the random encounter scenario. As can be observed from Fig. 2, up to a certain value of r the degree is larger with the network coding strategy than with GC. We shall observe that the encoding processes in these two cases also differ as in random linear coding the codewords are formed from a number of symbols in the node's memory and not by simply adding the node's own symbol. This has the effect of increasing the codeword diversity. Also, the degree curves intersect for a large number of recovered symbols as our analysis for the network coding case is based on the assumption of an infinite grid, whereas the Growth Code distribution has knowledge about the total number N of original symbols in the network. Including this knowledge in the analysis for the network coding case would lead to a similar increase of the degree as r approaches N.

We note that there is a substantial difference between network coding and GC in terms of how the degree is increased for an increasing r. This reflects the different scenarios (static *vs.* random encounter) for which the related distributions were designed. In addition, actual networks are likely to be neither completely static nor dynamic in the sense of [1]. For these intermediate cases, distributions lying between the static (network coding) and highly dynamic (Growth Codes) case might be a better choice. According to this intuition, in Fig. 2 we also plot the two heuristically derived distributions $r/5$ and $r/13$. These distributions will be considered for the simulation results presented in the remainder of this paper.

D. Modifications to Growth Codes

To improve the performance of GC in more general settings than the random encounter scenario in [1], we investigate the following modifications to the original GC algorithm:

1) **Multi-Codeword Encoding (MCE):** To increase the mixing of information within a neighborhood, we allow nodes to combine (i.e., `xor`) arbitrary codewords they have in memory. The encoding processes implemented by GC and MCE follow the same rules except for the choice of which information symbols have to be combined together. In both algorithms, a node at first randomly picks a codeword from its storage, c_1. According to GC, the node adds its own information unit to c_1 (if allowed by the degree distribution and if this information unit is not already included in c_1). With MCE, instead, the node randomly picks a further codeword c_2 from its memory and adds it to c_1. Subsequently, the node continues to add new codewords $c_3, c_4, \ldots$, all randomly picked from its storage and all different from each other, until the allowed codeword degree is reached. We stress that in static networks, or in networks with moderate mobility, adding the node's own symbol to all the codewords sent is of little use. In fact, this symbol is unlikely to be innovative (after its first transmission) when the node's neighborhood does not change over time. We finally observe that MCE simply states how codewords are to be combined together and, as such, can be used with any degree distribution.
2) **Gaussian Elimination (GE):** We further test whether using Gaussian elimination instead of the simpler decoder D described in [1] improves performance. In essence, decoder D only uses already decoded symbols for the decoding of the received information in a way similar to that of the message passing decoder in [11]. Gaussian elimination, instead, builds an encoding matrix according to the codewords locally stored in the node's memory and tries to invert it to retrieve the original symbols. Gaussian elimination is the optimal decoding method once the encoded codewords are given. Nevertheless, we shall observe that for large networks its complexity can be an issue due to the number of operations associated with matrix inversion. In [1], all the nodes store the received codeword in their own storage, while the sink node is the only device able to decode symbols. With Gaussian Elimination all nodes can locally decode as soon as their local encoding matrix has full rank.
3) **Aggressive degree distribution:** As indicated by the analysis in Section III-A, a more aggressive degree distribution is needed for optimal performance in static scenarios. In fact, GC increases the encoding degree rather slowly in order to give optimal performance in highly dynamic network settings. In the results reported in the following sections, we consider the degree distribution given by the heuristic $r/13$ in Fig. 2, which gave excellent results in less dynamic scenarios. We will use this distribution in order to obtain the performance of our new schemes. In addition, we will also see how GC performs when we replace its degree distribution with this heuristic.
4) **Decreased transmission probability for own information:** We compare the GC strategy, where the whole memory is initialized with the node's own symbol to a strategy where this symbol is stored only once. Accordingly, received codewords as well as the node's own symbol have the same probability of being picked for

transmission.

The above modifications were used to define a new algorithm and some of them were tested on GC to see whether it could be improved by means of slight changes. Specifically, with the term Modified Growth Codes (MGC), we mean the GC scheme where we consider Multi-Codeword Encoding (MCE) and Gaussian Elimination (GE) for the encoding and decoding, respectively. In the following sections MGC is compared against plain GC, which does not have these two features. In addition, both GC and MGC are studied for the standard (see Section III-B) codeword degree distribution (GC/MGC, standard) as well as the aggressive degree distribution (GC/MGC, aggressive), to isolate the impact of changing the codeword degree. For the aggressive distribution, in this paper we adopt the heuristic $r/13$ (see Fig. 2), where r is the cumulative number of recovered symbols at a node. We observe that a more in-depth analysis of suitable degree distributions is necessary to properly tune the codeword degree distribution according to the network dynamics. We leave this study for future research.

We further observe that the available memory at a node (i.e., the number of symbols it can store) has an impact on the diversity of the transmitted information. A large memory allows nodes to code over a more diverse set of codewords which makes it more likely to send innovative codewords when the neighbors already have most of the information available (i.e., after a number of transmissions have been performed). However, a large memory also increases the probability of initially (mainly) transmitting the node's own symbol, as GC initializes all the available memory with it. This can harm performance in more static settings, where the node's own symbol is unlikely to be innovative after its first transmission. In this respect, we note that increasing the maximum degree of the transmitted codewords beyond a certain threshold does not bring significant performance benefits in our simulation settings and we therefore limit it to 10 for all algorithms. This limits coding and decoding complexity as well as packet size. Note that, with this additional constraint the Gaussian elimination decoder can be implemented at a very low complexity and makes optimal decoding a viable solution for sensor devices. Also, the increase in complexity and memory consumption for all of the remaining modifications to GC that we propose here is comparatively low and the proposed algorithms can be implemented on resource constrained sensor hardware such as Crossbow motes.

IV. Simulation Results

Our reference scenario consists of a sensor network of $N = 100$ nodes, each having one information unit (sensor reading) to transmit via broadcast. As performance metrics, we analyze the average number of symbols recovered and the average codeword degree, that we plot as a function of the number of transmitted symbols (which roughly corresponds to simulation time). For the simulation, we consider three different settings, starting from a very dynamic scenario and ending with a static network:

- **Uniformly random encounters scenario:** Whenever two nodes meet, they exchange one codeword. Here, node neighborhoods are independent across subsequent observations. Growth Codes are specifically designed for this type of very dynamic (and sparse) network.
- **Mobile scenario:** Nodes move according to the random-waypoint mobility model. We consider a low ($2 \rightarrow 4$ m/s) as well as a high ($10 \rightarrow 20$ m/s) mobility scenario. The average node density is set to 4 or 8 neighbors per transmission range.
- **Static scenario:** Nodes are static and are placed on a grid network, again with 4 and 8 neighbors per transmission range.

The simulation results have been obtained using a custom C++ simulator. It provides an ideal (collision-free) MAC layer, with a sequential scheduling of tasks. It is noted that with GC the sink is the only node that can decode information symbols, while with MGC all nodes can potentially decode as well (if their storage allows).

A. Mixing Network Coding and Growth Codes

Next, in order to gain some understanding on the performance of the basic GC scheme, we compare it against random network coding and a mixture of network coding and Growth Codes. We consider a uniformly random encounters scenario. For network coding, we use a field size of two (i.e., packets are combined using simple `xor`), and each available codeword has a probability of 0.5 of being included in a new packet to be sent. With the mixed algorithm, the first packets are transmitted with GC. Then, after the number of received innovative packets reaches a certain threshold, the algorithm switches to network coding. Note that this might require additional information about the total number of symbols in the network (for a proper setting of the threshold).

The performance of the algorithms is shown in Fig. 3, where the term *number of codewords transmitted* indicates the total number of encoded packets sent per node (averaged over all nodes). The same metric will be considered for all remaining graphs. Notably, when switching from GC to network coding there is a sudden increase in the codeword degree, which prevents immediate decoding, until a sufficient number of high degree codewords have been received. For this reason, the graphs for network coding and the mixed algorithms show a flat phase immediately followed by a phase where the number of recovered symbols increases sharply. On average, standard GC takes around 250-300 transmitted codewords to recover all information, while the mixed codes only require between 110 and 200 codewords. The lower the switching threshold, the faster the codes achieve the maximum packet delivery ratio, but the lower the delivery ratio during the initial (flat) phase. The experiments show that using higher degree codewords particularly helps in the final part of the simulation. If we switch to higher degree codewords when 95% of the packets have been recovered, decoding performance is always above that of GC, resulting in a 50% reduction in the number of codewords required for full recovery. However, network coding mixes

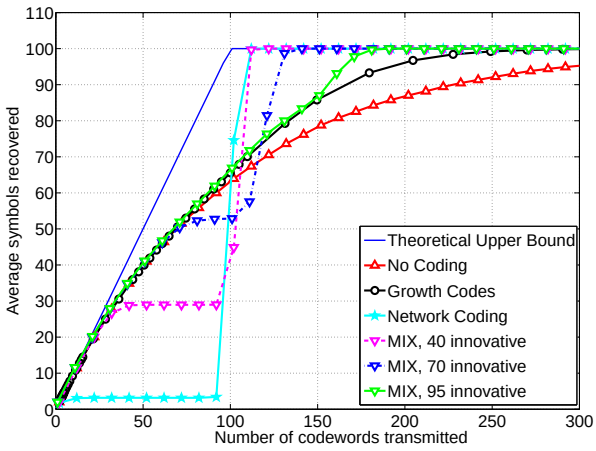


Fig. 3. Threshold-Based Mixed Codes compared to Growth Codes, No Coding and Network Coding, in the uniformly random encounters scenario.

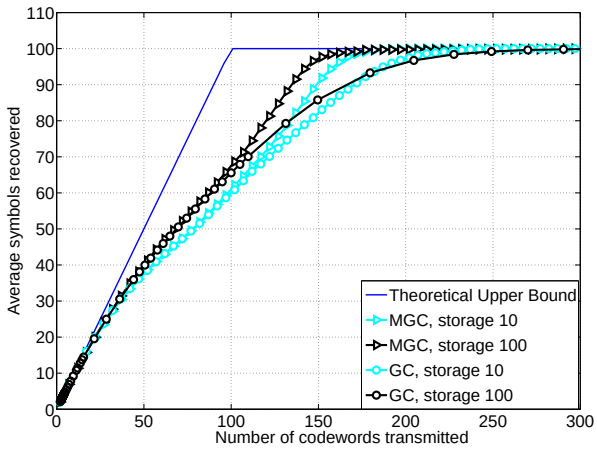


Fig. 4. Low storage size compared to large storage size for the standard degree distribution in the uniformly random encounters scenario.

	MGC	GC	Gain (%)
Storage 10, standard distrib	170	230	26
Storage 100, standard distrib	170	250	32

TABLE I
GAINS ACHIEVED BY MGC IN THE UNIFORMLY RANDOM ENCOUNTERS SCENARIO.

together a significantly larger number of packets to achieve this performance.

In the following sections, we analyze the light-weight modifications discussed in Section IV. We will start our investigation with the uniformly random encounters scenario, we will subsequently consider mobile networks with low and high mobility and, finally, we will present results for static networks.

B. Uniformly Random Encounters Scenario

We first compare GC and MGC in the uniformly random encounters scenario for which GC is designed. In this setting, the degree distribution of Growth Codes, referred to here as the *standard degree distribution*, is the best possible also for MGC. We will then consider the standard degree distribution for both GC and MGC for the results shown in this section. According to this distribution, nodes initially send uncoded symbols with high probability, then the degree is increased rather slowly up to a certain threshold and is finally increased sharply when almost all codewords have been recovered. It is noted that in the uniformly random encounters scenario Gaussian Elimination does not significantly improve performance, as the low degree of the codewords created by GC almost always allows immediate decoding with a message passing algorithm [11].

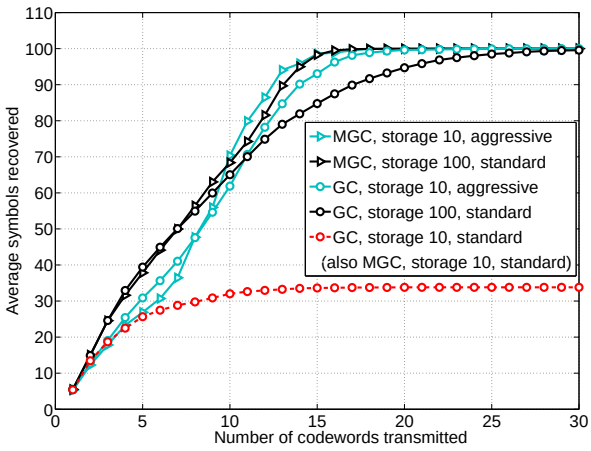
In Fig. 4, we compare GC and MGC for a large storage that can hold all available packets as well as for a smaller storage size of 10 codewords. With the small storage, both GC and MGC perform worse but the latter still gains over the former. A smaller storage size hinders a sufficient "mixing" of information and results in a small reduction of performance for intermediate numbers of recovered codewords. Nevertheless, the codes are able to achieve full recovery at roughly the same time as with unlimited storage. When storage space is scarce, a larger fraction of it will contain high degree codewords towards the end of the simulation, which increases the probability of sending out innovative codewords. In contrast, a very large buffer will still contain the node's own symbol a number of times, as well as many low degree codewords.

In Table I, we show the gains achieved by MGC in this scenario, in terms of number of transmitted codewords to recover all symbols. We remark that, since Gaussian Elimination is not that effective in this scenario, these gains are mainly due to the use of Multi-Codeword Encoding (MCE).

C. Mobile Scenario

Two different cases of random-waypoint mobility are considered: low mobility with minimum and maximum speeds of 2 m/s and 4 m/s, respectively, and high mobility with minimum and maximum speeds of 10 m/s and 20 m/s, respectively. Here, we only present results for the low mobility scenario with a node density of 8 neighbors per transmission range. The performance of the 4 neighbors case is similar to that of the high density scenario except that the curves are shifted to the right (as fewer nodes receive innovative information per packet transmission). In addition, results for the high mobility case are very similar to those obtained for the uniformly random encounters scenario.

First of all, we observe that with the standard degree distribution and a small storage size of 10 symbols, both GC and MGC are only able to decode around 30-40 symbols out of 100 (Fig. 5). This happens since with the standard degree distribution codewords initially have a low degree. In addition, the available storage is frequently overwritten due to the limited memory. Hence, information is lost faster than it can spread and this prevents completion of the recovery process. Also, in this case too few symbols are recovered to switch to higher degree codewords and the average degree of the transmitted codewords



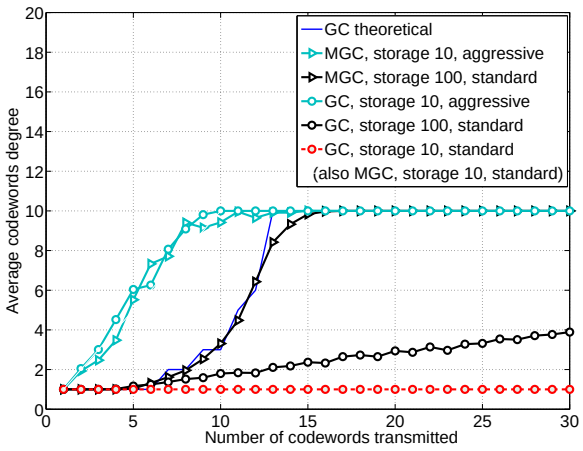


Fig. 5. Symbols recovered (left) and actual codewords degree (right) for MGC and GC in the mobile network setting with minimum speed 2 m/s and maximum speed 4 m/s, storage sizes of 10 and 100 packets, standard and aggressive degree distributions.

	MGC	GC	Gain (%)
Storage 10, aggressive distrib	17	20	15
Storage 100, standard distrib	17	30	43

TABLE II
GAINS ACHIEVED BY MGC IN THE LOW MOBILITY SCENARIO.

	MGC	GC	Gain (%)
Storage 10, aggressive distrib	20	100	80
Storage 100, standard distrib	70	230	70

TABLE III
GAINS ACHIEVED BY MGC IN THE STATIC NETWORK SCENARIO.

is very close to one, as shown in the right graph of Fig. 5. If the available storage is unlimited (e.g., ≥ 100 in this case), both GC and MGC recover all symbols, but MGC outperforms GC roughly by a factor of 2. This gain is mainly due to the use of Multi-Codeword Encoding (MCE). Again, GC is not able to achieve the theoretical distribution of codeword degrees, since the large buffer contains many low degree codewords and GC can only increase the degree by one by adding the node's own symbol. In contrast, the degrees of MGC closely match the desired distribution, as shown in the right graph of Fig. 5.

As expected, the aggressive degree distribution for the small storage case increases codeword degrees faster and thus decreases the performance gap between GC and MGC. More importantly, with the aggressive distribution, GC and MGC can both recover all symbols even with a small buffer of 10 packets. Moreover, although higher degree codewords are used, for all algorithms the number of codewords that cannot be immediately decoded is negligible. From these results, we can see that in scenarios that more closely reflect real-world network dynamics, the original GC has significant difficulties in achieving good performance when the node memory is limited. These problems are even more pronounced in static scenarios, for which GC admittedly was not designed. In Table II, we show the gains achieved by MGC in the low mobility scenario in terms of number of transmitted codewords needed for recovering all symbols.

D. Static Scenario

In the static topology case, as shown in Fig. 6, MGC substantially outperforms GC with both the aggressive and standard degree distributions. In case the neighborhood does not change across subsequent transmissions, sending the same uncoded information multiple times will obviously waste resources. Similarly, sending low degree codewords usually provides innovative information only to some of the neighbors. MGC with the aggressive distribution and small storage requires an order of magnitude fewer codeword transmissions than GC, even if GC is granted a much larger storage space. The performance of the original GC with storage 100 can however be improved. In particular, the performance of GC increases if all codewords in the storage have the same probability of being included in a transmitted codeword, as we show in Fig. 7. Nevertheless, the performance still remains well below that of the algorithms with our aggressive degree distribution. In Table III, we show the gains achieved by MGC in terms of transmitted codewords to recover all source symbols.

As expected, the standard GC degree distribution does not work for static scenarios. In this case a more aggressive distribution is instead more efficient as it increases the probability of transmitting innovative information at each transmission round. Also, Growth Codes profit less than MGC from using a more aggressive distribution and this is due to their coding process. In fact, always adding the node's own symbol to the codeword to send largely limits the achievable benefits.

We finally stress that, assuming a small storage size, the degree distribution should be aggressive in order to achieve good performance. This is particularly true for static scenarios with low density. What happens is that due to buffer limitations old symbols are discarded from the node's memory and, if no sufficient mixing of information is performed beforehand, they may be lost permanently. The aggressive distribution, together

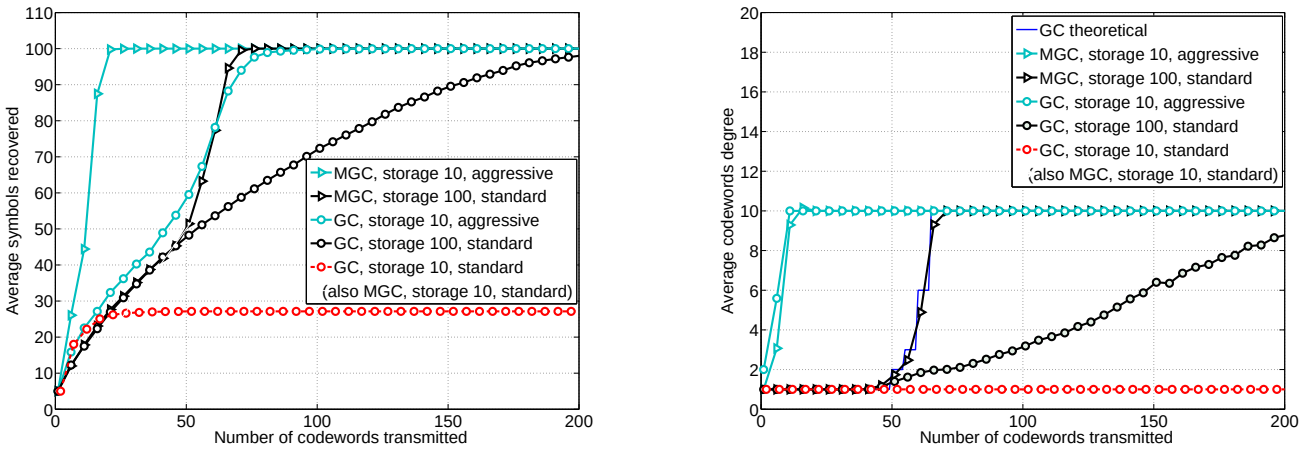


Fig. 6. Symbols recovered and degree distribution for MGC and GC in the static scenario. We analyze performance for storage sizes of 10 and 100 codewords and for both standard and aggressive degree distributions.

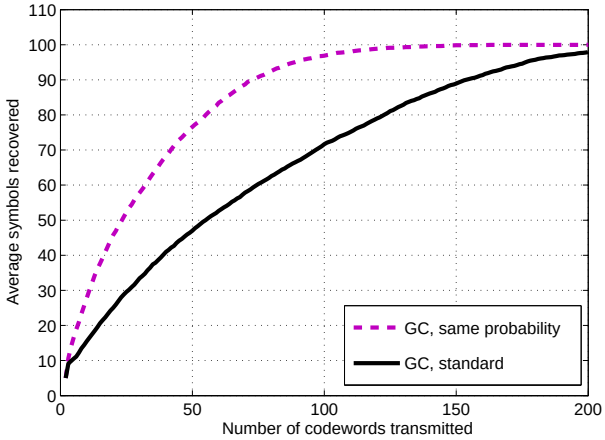


Fig. 7. Symbols recovered for the original GC and GC where all symbols have the same probability of being included in the transmitted codewords. Both algorithms use a standard degree distribution with a storage size of 100 codewords.

with our MCE encoding rule, ensures a proper mixing of the symbols in memory such that the information will survive even in the presence of memory limitations.

V. Conclusions

In this paper we proposed several modifications to the Growth Codes presented in [1], in order to increase their performance in static or mobile networks where the node's neighborhood varies slowly across subsequent transmission epochs. We proposed a new algorithm, named Modified Growth Codes (MGC), and we demonstrated its good performance. In particular, MGC was able to outperform standard Growth Codes in all simulation settings. The more static the scenario, the larger the performance gains. In our simulations we observed that when the buffer size is limited, standard GC could not achieve the intended codeword degree distribution due to the restrictions on how codewords are generated. This prevents a sufficient mixing of information and can substantially impact the performance.

While our algorithms show quite promising performance, they only give first insights into improving the design of Growth Codes. A more thorough analysis on how to adapt the codeword degree distribution to the specific dynamics of the network (and how to "measure" these dynamics) is necessary. An equally important avenue for future research is how to adjust the probability of including own information into the encoded symbols.

References

[1] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth Codes: Maximizing Sensor Network Data Persistence," in *ACM SIGCOMM*, Pisa, Italy, Sep. 2006.

[2] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[3] T. Ho, M. Medard, J. Shi, M. Effros, and D. R. Karger, "On Randomized Network Coding," in *41st Annual Allerton Conference on Communication Control and Computing*, Monticello, IL, US, Oct. 2003.

[4] S. Acedanski, S. Deb, M. Medard, and R. Koetter, "How good is random linear coding based distributed networked storage?" in *Netcod*, Riva Del Garda, Italy, Apr. 2005.

[5] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes," in *ACM/IEEE IPSN*, Nashville, CT, US, Apr. 2006.

[6] ——, "Decentralized Erasure Codes for Distributed Networked Storage," *IEEE/ACM Trans. on Networking*, vol. 14, no. SI, pp. 2809–2816, June 2006.

[7] ——, "Distributed Fountain Codes for Networked Storage," in *IEEE ICASSP*, Toulouse, France, May 2006.

[8] J. Widmer, C. Fragouli, and J.-Y. L. Boudec, "Low-complexity energy-efficient broadcasting in wireless ad-hoc networks using network coding," in *NetCod*, Riva del Garda, Italy, Apr. 2005.

[9] D. Wang, Q. Zhang, and J. Liu, "Partial Network Coding: Theory and Application in Continuous Sensor Data Collection," in *IWQoS*, Yale University, New Haven, CT, US, June 2006.

[10] P. A. Chou, T. Wu, and K. Jain, "Practical network coding," in *41st Allerton Conf. Communication, Control and Computing*, Monticello, IL, US, Oct. 2003.

[11] M. Luby, "LT Codes," in *43rd Ann. Symp. on Fundations of Computer Science*, Vancouver, Canada, Nov. 2002.