# A Perpetual Code for Network Coding

Heide, Janus; Pedersen, Morten Videbæk; Fitzek, Frank; Médard, Muriel

*Published in:*
IEEE Vehicular Technology Conference (VTC) - Wireless Networks and Security Symposium

*DOI (link to publication from Publisher):*
10.1109/VTCSpring.2014.7022790

*Publication date:*
2014

*Document Version*
Accepted author manuscript, peer reviewed version

# A Perpetual Code for Network Coding

Janus Heide, Morten V. Pedersen and Frank H.P. Fitzek
Faculty of Engineering and Science,
Aalborg University, Aalborg, Denmark
Email: [jah|mvp|ff]@es.aau.dk

Muriel Médard
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
Email: medard@mit.edu

***Abstract*—Random Linear Network Coding (RLNC) provides a theoretically efficient method for coding. The drawbacks associated with it are the complexity of the decoding and the overhead resulting from the coding vector. This adds to the overall energy consumption and is problematic for computational limited and battery driven platforms. In this work we present an approach to RLNC where the code is sparse and non-uniform. The sparsity allow for fast encoding and decoding, and the non-uniform protection of symbols enables recoding where the produced symbols are indistinguishable from those encoded at the source. The results show that the approach presented here provides a better trade-off between coding throughput and code overhead. In particular it can provide a coding overhead identical to RLNC but at significantly reduced computational complexity. It also allow for easy adjustment of this trade-off, which make it suitable for a broad range of platforms and applications. Finally it is easy to perform recoding and coding vectors can be efficiently represented.**

## I. Introduction

Network Coding (NC) is a promising paradigm [1] that have been shown to provide benefits in several of the existing network layers. NC enables coding at intermediate nodes in a communication network, and thus is fundamentally different from the end-to-end approach of channel and source coding. With NC data packets are no longer treated as atomic entities as they can be combined and re-combined at any node in the network. This allow for a less restricted view on the flow of information in networks, which can be particular helpful when building distribution systems for less structured networks such as meshed, peer-2-peer or highly mobile networks.

In this work we only consider random approaches to network coding RLNC [2], and disregard deterministic coding approaches as well physical layer techniques. The reason is that our primary focus is cooperative and highly mobile wireless networks, which fit perfectly with the highly decentralized nature of RLNC. In particular RLNC can be utilized to reduce the signaling overhead and increase robustness towards the changing channel conditions in the network. At the same time it allow for the construction of much simpler distribution systems, which from an engineering point of view is highly desirable. Unfortunately, RLNC is inherently computational demanding which have spawned several efforts to produce optimized implementations and modify the underlying code [3], [4]. Even though several solutions and implementations have been declared to provide *sufficient coding throughput* continued efforts are valid as they can enable NC on simpler devices, and reduce the energy consumption introduced by coding.

To ensure that the performance of RLNC is independent of the size of the data that is transmitted, the data is typically divided into *generations* [5]. This can reduce the computational complexity to a level that is practical usable and also reduce the decoding delay which is necessary for streaming applications. Unfortunately, it also increases the overhead of the code and introduces the need for additional signaling [6], [7]. To improve the trade-off between computational work and overhead it has been suggested to code over several generations [8] also called a *random annex* [9]. This reduces the problem of ensuring that all generations are decoded, and thus the overhead. At the same time it is less computational demanding as the decoding is performed in an inner and outer step. The approach is very useful for file transfers, but less so for streaming as the final decoding delay is high as generations are not decoded sequentially. Additional, the problem of how recoding could be performed has so far not been considered. In [10], [11] we considered some simplifications of the coding performed over each generation. Specifically, binary, systematic, and sparse codes, which can significantly increase the decoding throughput without introducing a high coding overhead. Unfortunately, the resulting codes are unsuitable for recoding, as explained in [11].

Here we continue this work and present a new code that is sparse, which allow for fast decoding, and has non-uniform protection of symbols enables for easy recoding. We introduce several new types of *recoding* and describe how these and *encoding* and *decoding* can be performed for the proposed code. Results obtained from an initial implementation show that the decoding complexity is low even at code overheads very close to that of RLNC. Additionally, the approach allow for easily adjusting this trade-off and hence it is applicable for a wide range of platforms. Finally it solves the practical problem of efficient coding vector representation, discussed in [11]. During our investigation we found an unpublished work [12] that present a very similar approach, and some more advanced variations. Therefore we have adopted their term *perpetual code*. We note that this approach and *random annex* approach are not mutual exclusive and could be combined.

The remainder of this paper is organized as follows. Section II introduces the coding operations encoding, decoding and different approaches to recoding. Section III provides performance analysis of the decoding complexity and code overhead, and compares with measurements obtained from our implementation. Final conclusions are drawn in Section V.

## II. Code Operation

This section introduces the code and the three operations, *encoding*, *decoding* and *recoding*, that can be performed at nodes in the network. The data to be transmitted from the source is divided into generations of size $g$, we denote the data in such a generation $\boldsymbol{M}$. Each generation is divided into packets that are combined as specified by a coding vector $\boldsymbol{g}$ over a Finite Field (FF) $\mathbb{F}_q$, and thus the code is linear, see [10], [13] for an introduction.

The elements in $\boldsymbol{g}$ are not drawn at random. Instead an element with index $p$ is chosen as the pivot, and the following $w$ elements are drawn at random from $\mathbb{F}_q$. We denote $w$ as the width of the coding vector. See Fig. 1 for a small example of the resulting coding vectors.

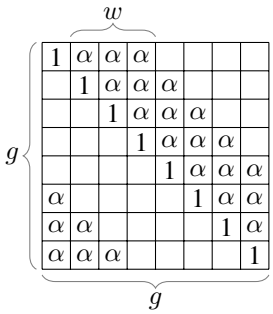



Fig. 1: All possible coding vectors, when $g = 8$ and $w = 3$. The $\alpha$'s denote different randomly drawn elements from $\mathbb{F}_q$.

The vectors are represented by an index and $w$ scalars and the necessary bits for their representation is thus given by Equation (1). The index can take $g$ values and each of each of the $w$ elements can take $q$ values.

$$|\boldsymbol{g}| = \log_2(g) + w \cdot \log_2(q) \text{ [bits]} \tag{1}$$

### A. Encoding

To encode a packet first a coding vector $\boldsymbol{g}$ is constructed. A randomly drawn index from the generation is used for the pivot, $p \in [0, g)$. This pivot element is set to one in $\boldsymbol{g}$. For the subsequent $w$ indices in $\boldsymbol{g}$ an element is drawn at random from $\mathbb{F}_q$. The remaining elements in $\boldsymbol{g}$ remain zero. To create a coded symbol the coding vector is multiplied onto the the data, $\boldsymbol{x} = \boldsymbol{M} \cdot \boldsymbol{g}$. Together the coding vector $\boldsymbol{g}$ and coded symbol $\boldsymbol{x}$ form a coded packet.

Encoding vectors can be generated in slightly different ways depending on how $p$ is drawn and the size of $w$, see Table I. Strictly speaking the *systematic* mode does not produce coding vectors of the specified form, but we include it for completeness.

TABLE I: Different encoding modes.

| Mode | $p$ drawn | $w$ |
|---|---|---|
| Random | at random from $[0, g)$ | $0 < w < g$ |
| Systematic | sequentially from $0-g, 0-g, \ldots$ | $w = 0$ |
| Pseudo-systematic | sequentially from $0-g, 0-g, \ldots$ | $0 < w < g$ |

### B. Decoding

As packets arrives at a node it places coded symbols in the data matrix $\hat{\boldsymbol{M}}$ and the coding vectors in the decoding matrix $\hat{\boldsymbol{G}}$. To decode the original data in $\hat{\boldsymbol{M}}$, $\hat{\boldsymbol{G}}$ must be brought onto identity form by performing basic row operations that is simultaneously performed on $\hat{\boldsymbol{M}}$. When it is not possible to decode a symbol fully upon reception, it is partially decoded, referred to as *on-the-fly* decoding, and stored for later processing. When enough symbols have been received so that $\hat{\boldsymbol{G}}$ has full rank, all received symbols can be fully decoded and the original data retrieved, we refer to this as *final* decoding.

*1) On-the-fly Decoding:* When a new coded packet arrives it is inserted into the decoding matrix iff. it has a pivot element that was not previously identified. Otherwise the previously received packet with the same pivot is subtracted from the new packet, and the pivot element of the new packet is changed. This is repeated until an new pivot element is identified for the received packet. If the packet is reduced to the zero vector, decoding terminates. It is possible to end in a dead-look where a sequence of rows is repeatedly subtracted from the new packet. To avoid this the decoding should be terminated after some attempts and the packet discarded. Our experiments show that by terminating after $2g$-$3g$ reductions most packets that can be decoded are decoded. To avoid wasting operations on such cases, row operations can first be performed on the coding vector and then repeated on the coded symbol.

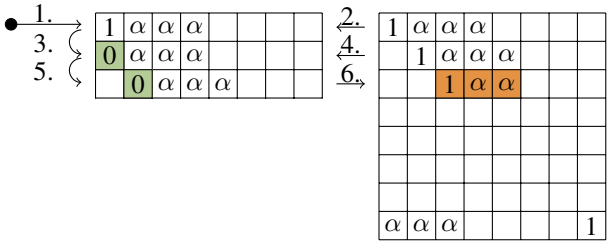



Fig. 2: *On-the-fly* decoding of a received coded packet. The right hand side matrix is the decoding matrix $\hat{\boldsymbol{G}}$. The left hand side matrix show the incoming packet as it is decoded. $\alpha$ denotes a random field element. The filled circle and arrow indicate the original incoming coded packet. The straight arrows indicate what rows are substituted into the received packet. The arching arrows indicate the step of the decoding of the received packet.

In Fig. 2, three coded packets have been received and inserted into the decoding matrix, the received packets have pivot element 0, 1, and 7 respectively. Subsequently a coded packet with pivot element 0 is received. This is denoted with a filled circle and arrow pointing to the packet in the left hand side matrix. A row has already been identified with the same pivot element as the incoming packet. Therefore the existing row 0 is subtracted from the incoming packet. This is denoted with the arrow pointing left into the right hand side matrix. The element that initially was the pivot element is now zero and an element to the right has now become the pivot element. This step is repeated for the new pivot and thus row 1 is subtracted from the incoming packet and element 2 becomes the pivot. As this pivot was previously not identified the row is inserted into the decoding matrix, which is marked with orange.

A special case is when the on-the-fly phase causes the pivot element to wrap around to the start of the coding vector. If the last element in the coding vector is reduced the first element in the vector is considered next and becomes the pivot element.

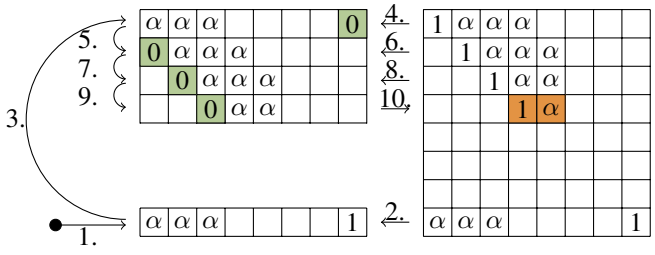
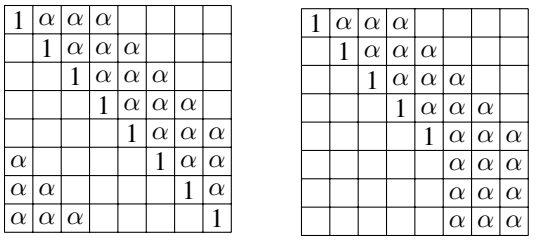



Fig. 3: Partial forward substitution into a received coded packet.

This is illustrated in Fig. 3. The incoming packet has pivot element 7 for which a pivot element have already been identified in $\hat{\boldsymbol{G}}$. Thus row 7 in $\hat{\boldsymbol{G}}$ is subtracted from the incoming packet. The resulting coding vector has a zero in index 7 and thus the pivot element is now index 0. The packet is then further reduced similar to the example in Fig. 2.

*2) Final Decoding:* When a pivot packet has been identified for all rows, final decoding is performed. This can be done using standard Gaussian elimination, we perform this in three simple steps, forward substitution, inversion and backwards substitution.



(a) The initial decoding matrix

(b) Decoding matrix after partial forward substitution

(c) Decoding matrix after inversion

(d) Decoding matrix after backwards substitution

Fig. 4: The decoding matrix $\hat{\boldsymbol{G}}$ at various states of the final decoding.

Initially the decoding matrix has a form similar to that shown in Fig. 4a. But the length of the vectors in all rows are not necessarily uniform and in that case the last element will not be monotonically increasing down through the rows. It should be noted that the inversion between Fig. 4b and Fig. 4c is not guaranteed to succeed in which case additional packets are needed.

### C. Recoding

During recoding non-decoded symbols are combined to create a new recoded symbol $\tilde{\boldsymbol{x}}$. This symbol is described by a coding vector $\tilde{\boldsymbol{g}}$. In classical RLNC this is performed as a separate operation and can result in a significant computational complexity, we denote this type of recoding *active* recoding. As explained in [11] this form of recoding is not suitable when the code is sparse, as the recoded symbol will be more dense with high probability. To combat these problems we introduce a new type of recoding and denote it *passive* recoding.

*1) Active Recoding:* When two or more coded or non-coded symbols have been received they can be combined by recoding. This is done by generating a local recoding vector $\boldsymbol{h}$ of length $g'$, where $g'$ is the number of received symbols. To create a recoded packet the collected coding vectors and coded symbols are combined as defined by $\boldsymbol{h}$. Thus $\tilde{\boldsymbol{x}} = \hat{\boldsymbol{M}} \cdot \boldsymbol{h}$ and $\tilde{\boldsymbol{g}} = \hat{\boldsymbol{G}} \cdot \boldsymbol{h}$ together form a recoded packet.

The elements in the resulting coding vector that can be non-zero is defined by the pivot elements and width $w$ of the rows used to recode. If packets with similar pivot elements are picked the resulting coded packet will in the worst case only have slightly more non-zero elements $w'$ than that of the original coding vectors. This decreases the freedom in recoding but allow us to maintain the sparsity in recoded packets. Furthermore, when such a recoded packet is decoded *on-the-fly* its width $w'$ will decrease back towards $w$.

*2) Passive Recoding:* When *on-the-fly* decoding has been performed, the received symbols which have been inserted as rows in $\hat{\boldsymbol{G}}$, are subtracted from the incoming symbol. This is done to decode the symbol but the operations can be reused for recoding and thereby reduce the computational complexity of recoding. If the operations performed on the symbols in $\hat{\boldsymbol{G}}$ are tracked, a symbol where a sufficient number of operations have been performed can be used as a recoded symbol.

It is necessary to track the progress to ensure that the recoded symbol is a combination of enough received symbols. One way is to keep a list for each received symbol, in which it is recorded what symbols are substituted into the symbol. However, if $g$ is high this could become unfeasible. It is simpler to hold an integer for each symbol that is used to count the number of other symbols that have been substituted into the symbol. During decoding we attempt to decode the symbols, therefore symbols that have been reduced *too much* should not be used as recoded symbols directly. We note that this *passive* approach can also be used for other codes.

*3) Active plus Passive Recoding:* To combine the two types of recoding we can monitor the passive recoding. If some neighboring set of packets combined meet our criteria for row operations, we can combine these by *actively* recoding them and thus obtain a recoded symbol. With this hybrid approach we can recode symbols whenever we need them and still reduce the computation work associated with recoding.

*4) Re-encoding:* When a receiver havs decoded a generation it can encode packets the same way as the original source. This is not recoding, and we call this re-encoding to distinguish this from encoding at the original source.

## III. Experiments and Analysis

In this section we present analytic and experimental results. All experimental data has been obtained with an initial implementation written in Python. For each setting 1000 runs was performed where coded packets from the encoder was fed to the decoder until it successfully decoded.

We are interested in exploring the overhead and decoding complexity. The overhead is the expected ratio of extra packets that must be received by the decoder to obtain full rank and thus successfully decode. The decoding complexity is evaluated as the number of row operations performed by the decoder in order to complete the decoding procedure. We express the computational complexity in the compound metric row *multiplication-addition*, where a multiplication-addition is multiplying a row with a scalar and adding or subtracting it to or from another row. Here we only consider the binary field therefore the multiplication scalar is always one, and a row multiplication-addition is simply adding or subtracting a row to or from another row.

### A. Coding Complexity

To *encode* a single packet, the expected number of row operations is given by Equation (2). We start with an empty vector and first add the chosen pivot row to it. For each of the following $w$ rows that row is multiplied with a random element from $\mathbb{F}_q$ and added to the new row. The probability that a randomly drawn element from $\mathbb{F}_q$ with size $q$ is non-zero is $1 - \frac{1}{q}$.

$$1 + w \cdot (1 - \frac{1}{q}) \tag{2}$$

From our experiments we have determined an empirical expression for the number of row operations performed during the *on-the-fly* phase of decoding for a whole generation denoted $\delta_{\text{fly}}$, see Equation (3). This was obtained under the condition that $w$ is sufficiently high, meaning that the code overhead is approaching that of traditional RLNC, see Section III-B.

$$\delta_{\text{fly}} \approx e^{\log_2(g)} \cdot (1 - q^{-1}) \tag{3}$$

To obtain an upper bound for the *final decoding* we consider the worst case, where most scalars are non-zero, see Fig. 4a. First we bring the bottom $w$ rows on pivot form. Therefore the top $g - w$ rows is substituted into the $w$ bottom rows which brings the decoding matrix to the form in Fig. 4b. The upper bound on the total number of operations is therefore defined by Equation (4). Then the bottom $w$ rows are brought onto echelon form, see Fig. 4c. The $(g - w)$'th rows is substituted into the below $(w-1)$ rows, the $(g-w+1)$'th row is substituted into the below $(w - 2)$ rows and so on, which is equal to the sum in Equation (5).

$$\delta_{\text{sub}} \leq (g - w) \cdot w \tag{4}$$

$$\delta_{\text{inv}} \leq \sum_{i=1}^{w-1} i = \frac{w \cdot (w - 1)}{2} \tag{5}$$

To finalize the decoding a similar procedure is performed, but this time upwards. Thus the number of operations is exactly the same as in Equation (4) and Equation (5). We divide by $g$ as $\delta$ is defined as the expected operations per packet. To include the probability that an element in $\mathbb{F}_q$ is equal to zero, we multiply $(1 - q^{-1})$ onto the on-the-fly, substitution, and inversion phases and obtain Equation (6).

$$\begin{aligned} \delta &\leq (\delta_{\text{fly}} + 2\delta_{\text{sub}} + 2\delta_{\text{inv}})/g \\ &= \left(\frac{1}{g}\right)\left(1 - \frac{1}{q}\right)\left(e^{\log_2(g)} + w \cdot (2g - w - 1)\right) \end{aligned} \tag{6}$$

In Fig. 5 the number of row multiplication-additions performed to decode one generation, both during the *on-the-fly* and *final* decoding phase, is plotted for different values of $g$ and $w$. The generation size and width is noted on the x-axis, and the number of row operations per decoded packet is on the y-axis. The operations during *on-the-fly* and *final* decoding is stacked to also show the total number of row operations.

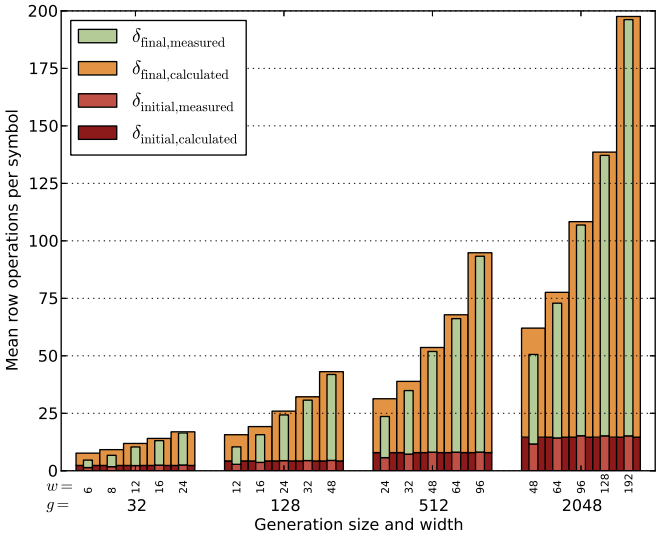



Fig. 5: Mean row operations per decoded symbol

Both the expression for the *on-the-fly* and *final* decoding fit the measurements, especially when $w$ is sufficiently high. For low values of $w$ the bound is less tight, but such settings should not be used when the code overhead is considered.

We compare the obtained results with traditional RLNC where the expected operations used to decode a packet is approximately $g/2$ for the binary case [14]. The factor between total operations used to decode for RLNC and the perpetual approach is calculated as $\delta_{\text{RLNC}}/\delta_{\text{Perpetual}}$, see Table II. In the best case $g = 2048$ and $w = 96$, RLNC is almost ten times more computational demanding than the proposed approach.

TABLE II: Factor of complexity compared to RLNC

| $g$ | 32 | | | 128 | | | 512 | | | 2048 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $w$ | 12 | 16 | 24 | 24 | 32 | 48 | 48 | 64 | 96 | 96 | 128 | 192 |
| Factor | 1.5 | 1.2 | 1.0 | 2.6 | 2.1 | 1.5 | 4.9 | 3.9 | 2.7 | **9.6** | 7.5 | 5.2 |

### B. Code Overhead

The code overhead specifies the amount of additional data must be transmitted in addition to the original data and depends on the field size, density, generation size but also on the entropy of the coding. Hence a structured code where symbols are not just combined at random will have a higher overhead compared to a completely random with the same coding parameters. From standard RLNC we know the lower bound for the code overhead as defined in Equation (7), see [11]. The same lower bound holds here, as the lowest overhead is obtained when $w$ is as high as possible, in which case the perpetual code becomes identical to RLNC.

The lower bound in Equation (7) evaluate the expected overhead based on the probability that the rank is increased at the receiver when a new coded symbol is received. This is a function of the generation size, $g$, the field size $q$, and the rank at the receiver, $g'$. For each of the indices where the decoder have already identified a pivot element, the index in the incoming packet is reduced to zero by the decoder. Hence the remaining $g - g'$ can in the best case be considered as elements drawn at random from $\mathbb{F}_q$. Hence the probability that these are all zero and the packet is linear dependent is $1/q^{g-g'}$. The mean overhead is then calculated as the sum of the expected amount of overhead for the decoding of each packet, for all possible ranks of the decoder. Note that the overhead is primarily due to the last packets, and that it become negligible for high values of $q$.

$$\begin{aligned}\alpha &\geq \sum_{g'=0}^{g-1} \left( \left( 1 - \frac{1}{q^{g-g'}} \right)^{-1} - 1 \right) \\ &= \sum_{g'=0}^{g-1} \left( \frac{1}{q^{g-g'} - 1} \right)\end{aligned} \tag{7}$$

For a symbol to be independent, either its pivot or one of the $w$ coefficient must hit a new pivot element. The pivot of the symbol is independent and hence the probability is $\frac{1}{g}$, but the $w$ elements depend on the pivot. The probability that one of these $w$ elements hits an uncovered pivot is $\frac{r'}{g}$ where $r' = [1, g-1]$. The expected number of tries to hit an unseen pivot is thus.

$$\sum_{r'=g}^{1} \left( \frac{r'}{g} \right)^{-1} = g \cdot \sum_{r=0}^{g-1} \frac{1}{g-r} \tag{8}$$

Hence the probability that one of the $w$ elements hits an unseen pivot can be expressed as $w / \left( g \cdot \sum_{r=0}^{g-1} \frac{1}{g-r} \right)$. Then the probability that a symbol is covered when $x$ coded symbols have been received can be found as the probability that none of the $x$ coded symbols covers the symbol. In the worst case, decoding is possible when all $g$ pivots are covered.

$$F_X(x) \geq \left( 1 - \left( 1 - \left( \frac{1}{g} + w / \left( g \cdot \sum_{r=0}^{g-1} \frac{1}{g-r} \right) \right) \right)^{x} \right)^{g} \tag{9}$$

The resulting *cdf* can be used to calculate an upper bound for the code overhead by evaluating the corresponding *survival function (sf)*, which defines the probability that there is an uncovered symbol after $x$ transmissions and thus additional transmissions are necessary.

$$\begin{aligned}\beta &\leq \sum_{x=g}^{\infty} S_X(x) = \sum_{x=g}^{\infty} 1 - F_X(x) \\ \alpha &\leq O \leq \alpha + \beta\end{aligned} \tag{10}$$

In Fig. 6 the overhead for different generation sizes is plotted as a function of the width. The width of the used code is shown on the x-axis. On the y-axis is the resulting overhead given in percent. The dotted lines denote the bounds.

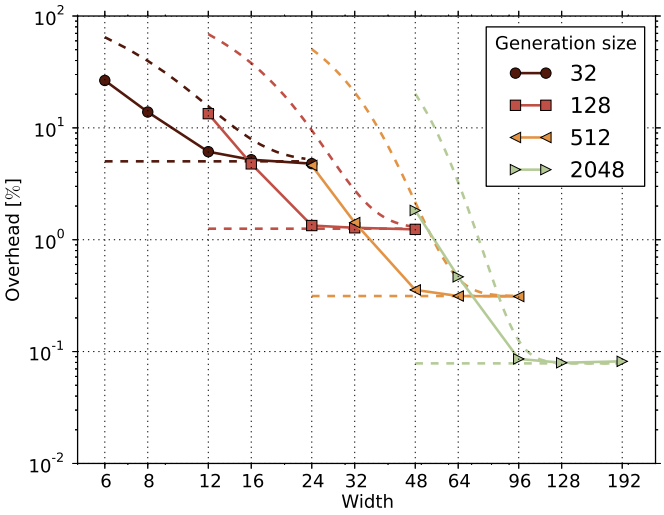



Fig. 6: Code overhead in percent as a function of $g$ and $w$. The dotted lines corresponds to the bounds for the overhead for a given generation size.

For each generation size, the overhead decreases as the width increases until the width is sufficiently high and the overhead becomes indistinguishable from the lower bound. If the width is decreased below the sufficient level, the overhead increases significantly. Therefore, values of $w$ below this point should generally not be used. The bounds are loose for low values of $w$, but become tighter as $w$ increases. Thus the provided bounds are useful for identifying a value of $w$ that is sufficiently high.

To make a comparison with RLNC we must consider Fig. 5 and Fig. 6 at the same time, as the performance of the codes are a trade-off between code complexity and code overhead. As the lower bounds are the same as for RLNC we can never hope to achieve a lower code overhead. However, we can achieve the same overhead but at lower computational complexity, see Fig. 5. The values of $w$ chosen in Table II corresponds to the cases where the code overhead of RLNC and the perpetual code are similar. Thus this approach can deliver a similar code overhead as RLNC at significantly decreased decoding complexity. For the reported settings this gain is up to a factor of ten.

## IV. Discussion

We have only considered the *random* encoding mode, thus the pivot element is always drawn at random and independently of the previous pivots, see Table I. This corresponds to the worst-case, where the channel is extremely lossy and thus systematic approaches are of no benefit. In cases where the erasure probability is low or moderate, a systematic or pseudo-systematic mode could be used which would decrease the code overhead and in particular the decoding complexity.

Ideally all decoding should be performed *on-the-fly* as this decreases the final decoding delay and distributes the processing load evenly. At the same time decoding should be performed in such a way that *fill-in* [15] does not occur, as this reduces the amount of work necessary to decode. In our presented results the ratio of operations performed in the *on-the-fly* phase is low, see Fig. 5. Fortunately the structure of the code makes it possible to perform what that can best be described as opportunistic backwards substitution. Our tests with this approach show that most of the decoding operations can be performed when symbols are received. However, this algorithm is sufficiently more complicated to analyze and due to space constraint we have omitted it.

The approach presented here is similar to what is called a *smooth perpetual code* in [12], but with two significant differences, we do not use zero padding nor a precode. Since zero padding is not used the overhead of the code is reduced as all original symbols are represented with equal probability. However, it also complicates the final decoding step. By not using a precode the code becomes simpler to analyze and implement, but it also increases the overhead of the code. As our interest is towards practical implementation we believe that our choice is sound and that it is reasonable to first consider the simpler case and add complexity later. Especially as we are interested in very computational constrained platforms where it might not be possible to use the more complex features.

Finally we note that *perpetual codes* are not a substitution but a supplement to RLNC. Specifically we believe that RLNC is a good choice for low generation sizes, but perpetual codes are more suitable at medium sized generations and possible generations sizes similar to those of fountain codes [16].

## V. Conclusion

In this paper we presented our initial findings on *perpetual codes* which is a new take on the fundamental RLNC approach. We described how encoding, decoding, and recoding can be performed and provided initial analysis of the codes performance in terms of code overhead, and coding complexity. This is compared with measurements obtained with an initial Python implementation. The results show that the proposed approach can significantly improve the trade-off between computational complexity and code overhead. Specifically the a coding overhead is similar to RLNC but at a much lower computational complexity, a complexity reduction of up to a factor of ten was measured. Additionally the approach provides an easily adjustable parameter that allow the trade-off to be tweaked to the used platform and application.

For the future we hope that a better understanding of the *on-the-fly* phase can help to provide a thorough analysis of the code. This knowledge is also a necessary condition for understanding the impact of the additions proposed in [12]. Currently we are working on an optimized C++ implementation which is needed to determine the throughput rate. As the algorithms are simple we are confident that our implementation will be fast. Additionally, we are considering advanced decoding algorithms in order to reduce the final decoding delay even further.

## Acknowledgment

This work was partially financed by the CONE project (Grant No. 09-066549/FTP) granted by the Danish Ministry of Science, Technology and Innovation.

## References

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

[2] T. Ho, R. Koetter, M. Médard, D. Karger, and M. ros, "The benefits of coding over routing in a randomized setting," in *Proceedings of the IEEE International Symposium on Information Theory, ISIT '03*, June 29 - July 4 2003. [Online]. Available: citeseer.ist.psu.edu/ho03benefits.html

[3] S. Yang and R. W. Yeung, "Large file transmission in network-coded networks with packet loss: a performance perspective," in *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, ser. ISABEL '11. Barcelona, Spain: ACM, 2011, pp. 117:1–117:5. [Online]. Available: http://doi.acm.org/10.1145/2093698.2093815

[4] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, *Network Coding in the Real World*. Academic Press, October 2011, ch. 4, pp. 87–114.

[5] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," *Proceedings of the annual Allerton conference on communication control and computing*, vol. 4, pp. 40–49, 2003.

[6] P. Maymounkov, N. J. A. Harvey, and D. S. Lun, "Methods for Efficient Network Coding," *44th Allerton Annual Conference*, 2006.

[7] Y. Li, E. Soljanin, and P. Spasojevic and, "Collecting coded coupons over overlapping generations," in *Network Coding (NetCod), 2010 IEEE International Symposium on*, june 2010, pp. 1 –6.

[8] D. Silva, W. Zeng, and F. Kschischang, "Sparse network coding with overlapping classes," in *Network Coding, Theory, and Applications, 2009. NetCod '09. Workshop on*, june 2009, pp. 74 –79.

[9] Y. Li, E. Soljanin, and P. Spasojevic, "Effects of the generation size and overlap on throughput and complexity in randomized linear network coding," *CoRR*, vol. abs/1011.3498, 2010.

[10] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, "Network coding for mobile devices - systematic binary random rateless codes," in *The IEEE International Conference on Communications (ICC)*, Dresden, Germany, 14-18 June 2009.

[11] J. Heide, M. V. Pedersen, F. H. Fitzek, and M. Médard, "On code parameters and coding vector representation for practical rlnc," in *IEEE International Conference on Communications (ICC) - Communication Theory Symposium*, Kyoto, Japan, jun 2011.

[12] P. Maymounkov. (2009) Perpetual codes: cache-friendly coding. Unpublished draft, retieved 2nd of September 2011. [Online]. Available: http://pdos.csail.mit.edu/~petar/papers/maymounkov-perpetual.ps

[13] C. Fragouli, J. Boudec, and J. Widmer, "Network coding: an instant primer," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 63–68, 2006.

[14] J. Heide, M. V. Pedersen, and F. H. Fitzek, "Decoding algorithms for random linear network codes," in *IFIP International Conferences on Networking - Workshop on Network Coding Applications and Protocols (NC-Pro)*, ser. Lecture Notes in Computer Science, vol. 6827, Valencia, Spain, may 2011, pp. 129–137.

[15] S. Ingram. (2006) Minimum degree reordering algorithms: A tutorial. Retrieved March 2010.

[16] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.