Intrinsic Information Conveying for Network Coding Systems

Heide, Janus; Pedersen, Morten Videbæk; Fitzek, Frank; Zhang, Qi

Published in:
I E E E V T S Vehicular Technology Conference. Proceedings

DOI (link to publication from Publisher):
10.1109/VETECF.2011.6093005

Publication date:
2011

Document Version
Early version, also known as pre-print

Intrinsic Information Conveying for Network Coding Systems

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

Qi Zhang
Aarhus school of Engineering, Aarhus, Denmark
Email: qz@cs.au.dk

***Abstract*—This paper investigated the possibility of intrinsic information conveying in network coding systems. The information is embedded into the coding vector by constructing the vector based on a set of predefined rules. This information can subsequently be retrieved by any receiver. The starting point is Random Linear Network Coding (RLNC) and the goal is to reduce the amount of coding operations both at the coding and decoding node, and at the same time remove the need for dedicated signaling messages. In a traditional RLNC system, coding operation takes up significant computational resources and adds to the overall energy consumption, which is particular problematic for mobile battery-driven devices. In RLNC coding is performed over a Finite Field (FF). We propose to divide this field into sub fields, and let each sub field signify some information or state. In order to embed the information correctly the coding operations must be performed in a particular way, which we introduce. Finally we evaluate the suggested system and find that the amount of coding can be significantly reduced both at nodes that recode and decode.**

I. INTRODUCTION

Network Coding (NC) is a promising concept that breaks with the existing store-and-forward paradigm in computer networks, and has been shown to achieve capacity in any communication network [1]. NC breaks with the end-to-end approach of channel and source coding, as packets are no longer treated as atomic entities, and data may be combined and re-combined at any point in the network. This new feature can provide advantages over traditional routing in meshed networks, and be a powerful tool in mobile multi-hop communication systems.

As an example consider the following cooperative scenario where several mobile devices wish to receive the same data. Each node is connected to a global overlay network that it receives data from, such as UMTS or LTE. In addition the nodes are in close proximity and connected locally via WiFi. The connection among the nodes can be direct links or realized by relaying. If the mobile nodes want to share their data, the mobile nodes can form a cluster where they exchange data locally. NC can be very useful for the local exchange as it can be used to overcome the *coupon collector's* problem [2]. However, one remaining problem is how the packets should be coded to ensure efficient cooperation.

With the COPE method introduced in [3], each node attempts to combine packets based on what packets the neighboring nodes need. When a node sends a packet it attempts to identify a set of packets that its neighboring node needs, code these packets together, and send the resulting coded packet. Thus the sender can satisfy all receivers with one coded packet. In broadcast systems this can only be achieved if all nodes needs the same packet, otherwise the sent packet will be useless for some of the receivers. One problem with this approach is that all nodes must obtain knowledge about what have been received by the other nodes in the cluster. In order to obtain this knowledge some signaling may be necessary which introduces overhead.

RLNC was introduced in [4] to remove the need to gather information about neighboring nodes. With RLNC packets are coded "randomly" and if the used FF size is large enough, the probability of generating linear dependent packets is small. On the downside the computational complexity is much higher in RLNC compared to COPE. Furthermore RLNC requires some mechanism that determines when the nodes have received enough packets. For instance, in the described cooperative scenario, the nodes must know when to stop the local exchange of packets, this require some form of signaling.

In RLNC a coded packet is a combination of all the packets available at the coding node. The performed coding is described by the coding vector. For RLNC this coding is dense, as many packets are combined, and therefore the computational complexity of the coding is high. If fewer packets are combined, the density of the coding vector decreases, it becomes more sparse. This, which decreases the computational complexity of both encoding and decoding the packet. However, if it is not done carefully it can increase the amount of linear dependent packets created. See [5] for an overview of gossip approaches to reduce the coding complexity.

Therefore we advocate exploiting the coding vector to gain knowledge about the packets received by neighboring nodes and to identify when the cooperative exchange can be stopped. Thus this necessary information can be distributed in the cluster without additional signaling. To achieve this we propose to dynamically and intelligently craft the coding vectors based on the information available at the coding node.

The remainder of this paper is organized as follows; Section II explains how information can be embedded into the coding vector, and introduces an example of information that can be embedded. In Section III we consider a cooperative network topology and compare the amount of coding when RLNC is used alone and in combination with conveyed intrinsic information. The final conclusions are drawn in Section IV.

II. Dividing the Field

All coding operations are performed over a Finite Field (FF), of size q, and thus the original data is represented by a series of $\lceil \frac{m}{q} \rceil$ field elements, each of size q. In the same way each coding vector is represented by g field elements of size q, where each element in the coding vector describes the operations performed on the corresponding symbol. Typically the elements in the coding vector are drawn at random from q. Instead we propose to divide this field into n sets $A_1, A_2, \ldots, A_n$, where $q = |A_1| + |A_2| + \ldots + |A_n|$. Each subset can be associated with some condition at the coding node, and thus be used to embed information into the coding vector.

We use the following rules to illustrate the idea; a field element in set A_1 indicates that a pivot element has been identified for the corresponding symbol at the sender, and a field element in set A_2 indicates that no pivot element has been identified for that symbol. An additional set $A_0 = 0$ indicates nothing and is necessary for reasons we will return to. For a sink each of the g symbols in a generation can be in one of three states, *unknown*, *no-pivot*, or *pivot*. All sinks hold this state information for each of the other sinks. Whenever a sink receives a coded symbol it updates the g states that corresponds to the sender, where transitions between the states occur as illustrated on Figure 1.

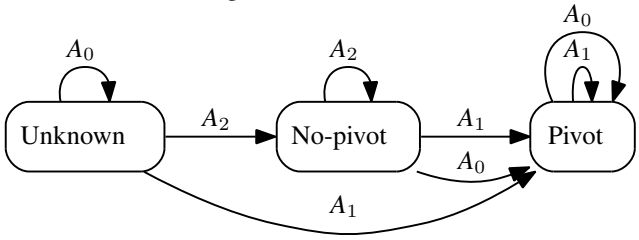

Fig. 1: The three states and the possible transitions.

If a sink has not yet identified a pivot element for a symbol, it needs a coded symbol that includes that particular symbol to complete the decoding. Therefore indicating *no-pivot* for a symbol can be interpreted as a request for that symbol. When a node identifies a pivot element for a symbol, it may omit that from future coding vectors to create more sparse packets. This is done by setting the corresponding element to 0 in the coding vector. Therefore a *no-pivot* indication followed by a 0 A_0 indicate that a pivot has been identified. If pivot elements for all symbols are available among the sinks in the cluster, they will be able to decode the original data by exchanging symbols. Thus if the sinks indicate for which symbols they have a pivot element, this can be used to determine when the entire cluster holds enough symbols to decode. Additionally any receiver can determine the rank at the sender, simply by counting the number of indicated pivot elements.

The reason we do not convey more precise information such as, *symbol decoded*, and *symbol not decoded*, should become apparent when we explain how this information can be embedded when a symbol is coded. To understand how the information can be embedded we need to take a closer look at the available coding operations, *encoding*, *decoding*, and *recoding*. Readers unfamiliar with NC can refer to [6] for an introduction. Data to be transferred from the source to the sinks is divided into packets of length m. The number of original packets over which encoding is performed is typically referred to as the batch size or generation size and denoted g. Thus the g original data packets of length m are arranged in the matrix $\boldsymbol{M} = [\boldsymbol{m}_1; \boldsymbol{m}_2; \ldots; \boldsymbol{m}_g]$, where $\boldsymbol{m}_i$ is a column vector. When a coded symbol is transmitted into the network, it must be accompanied by a coding vector that describes the operations performed to create the coded symbol. The coding vector is used to decode or recode the symbol at other nodes in the network that receive the symbol.

A. Encoding

Normally to encode a packet $\boldsymbol{x}$ at the source, $\boldsymbol{M}$ is multiplied with a randomly generated coding column vector $\boldsymbol{g}$ of length g, $\boldsymbol{x} = \boldsymbol{M} \times \boldsymbol{g}$. In this way we can construct $\boldsymbol{X} = [\boldsymbol{x}_1; \boldsymbol{x}_2; \ldots; \boldsymbol{x}_{g+r}]$ that consists of $g + r$ coded data packets and $\boldsymbol{G} = [\boldsymbol{g}_1; \boldsymbol{g}_2; \ldots; \boldsymbol{g}_{g+r}]$ that contains $g + r$ randomly generated encoding vectors, where r is any number of redundant packets.

In order to embed information into the coding vector, $\boldsymbol{g}$ is not drawn randomly but instead, from one of the sets A_i based on the defined conditions. With the suggested approach all field elements for coding vectors generated at a source are in A_1. To allow sources to encode sparse packets, $0 \notin A_2$, as otherwise 0 would incorrectly indicate that the source has no pivot element for that symbol. Note that when encoding all elements in $\boldsymbol{g}$ can be chosen arbitrarily because the source holds all original symbols, and thus all possible $\boldsymbol{g}$ are valid encoding vectors. If a coding vector consists of all 0's except a single element that is 1, the coded packet is equal to an original symbol and we say that it is trivially coded.

B. Decoding

When a coded symbol is received the embedded information of the coding vector is first retrieved. The goal of decoding is to transform the received coded symbols into the original symbols and thus obtain the original data $\boldsymbol{M}$. To complete the decoding, g linear independent coded symbols and coding vectors are needed. Decoding should be performed on-the-fly in order to distribute the computational work and determine the progress of the decoding. During decoding, it is more convenient to consider the coded symbols and coding vectors as row vectors. Thus all g' received symbols are collected in $\hat{\boldsymbol{X}}^\mathsf{T}[\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_{g'}]$ where $\hat{\boldsymbol{x}}_i$ is a coded symbol. And the corresponding coding vectors are collected in $\hat{\boldsymbol{G}}^\mathsf{T} = [\hat{\boldsymbol{g}}_1, \hat{\boldsymbol{g}}_2, \ldots, \hat{\boldsymbol{g}}_{g'}]$ where $\hat{\boldsymbol{g}}_i$ is a coding vector. We denote $\hat{\boldsymbol{G}}^\mathsf{T}$ the decoding matrix as it holds the information necessary to decode the received symbols in $\hat{\boldsymbol{X}}^\mathsf{T}$. To decode the original data, $\hat{\boldsymbol{G}}^\mathsf{T}$ is transformed into the identity matrix, by performing row operations, that is simultaneously performed on $\hat{\boldsymbol{X}}^\mathsf{T}$. In this way $\hat{\boldsymbol{X}}^\mathsf{T} \rightarrow \boldsymbol{M}^\mathsf{T}$ as $\hat{\boldsymbol{G}}^\mathsf{T} \rightarrow \boldsymbol{I}$.

Equation (1) is an example of a part of a decoding matrix, $\hat{\boldsymbol{G}}^\mathsf{T}$, for a node that has received four linear independent

symbols. In the attempt to bring $\hat{\boldsymbol{G}}^{\mathsf{T}}$ to a reduced echelon form, pivot elements have been identified for the indices 0,1,2, and 4. No pivot element has been identified for index 3, thus no coding vector has been inserted into the corresponding rows. Additionally $\hat{g}_{5,3} = 0$, as if this was not the case a pivot element would have been identified for index 3. The remainder of rows 0,1,2, and 4 can be any field elements.

$$\hat{\boldsymbol{G}}^{\mathsf{T}} = \begin{bmatrix} \mathbf{1} & 0 & 0 & \hat{g}_{3,0} & 0 & \hat{g}_{5,0} & \cdots \\ 0 & \mathbf{1} & 0 & \hat{g}_{3,1} & 0 & \hat{g}_{5,1} & \cdots \\ 0 & 0 & \mathbf{1} & \hat{g}_{3,2} & 0 & \hat{g}_{5,2} & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & \mathbf{1} & \hat{g}_{5,4} & \cdots \end{bmatrix} \tag{1}$$

C. Recoding

Recoding is similar to encoding so we represent the symbols and coding vectors as column vectors. It is more complicated to embed information into the coding vector during recoding compared to encoding. The reason is that the vector used to recode, we denote this $\boldsymbol{h}$, is not the vector that is transmitted together with the resulting coded symbol. Any node that has received $g' > 1$ linear independent packets, can recode and thus create new coded symbols. All received g' symbols are held in the matrix $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1 \hat{\boldsymbol{x}}_2 \dots \hat{\boldsymbol{x}}_{g'}]$ and all coding vectors are in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1 \hat{\boldsymbol{g}}_2 \dots \hat{\boldsymbol{g}}_{g'}]$. To recode a symbol $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{X}}$ are multiplied with a randomly generated vector $\boldsymbol{h}$ of length g', $\tilde{\boldsymbol{g}} = \hat{\boldsymbol{G}} \times \boldsymbol{h}$, $\tilde{\boldsymbol{x}} = \hat{\boldsymbol{X}} \times \boldsymbol{h}$. Note that $\boldsymbol{h}$ is only used locally and that there is no need to distinguish between coded and recoded symbols.

We reuse the example from Equation (1) to compute a new coding vector, $\tilde{\boldsymbol{g}}$, to illustrate the problem. Note that any $\boldsymbol{h}$ is valid as long as $h_i = 0$ for every index where no pivot element has been identified in $\hat{\boldsymbol{G}}^{\mathsf{T}}$.

$$\overbrace{\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ \hat{g}_{3,0} & \hat{g}_{3,1} & \hat{g}_{3,2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} \\ \hat{g}_{5,0} & \hat{g}_{5,1} & \hat{g}_{5,2} & 0 & \hat{g}_{5,4} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}}^{\hat{\boldsymbol{G}}} \times \overbrace{\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ 0 \\ h_4 \end{bmatrix}}^{\boldsymbol{h}} = \overbrace{\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \sigma_3 \\ h_4 \\ \sigma_5 \\ \vdots \end{bmatrix}}^{\tilde{\boldsymbol{g}}} \tag{2}$$

$$\text{where} : \sigma_j = h_0 \cdot \hat{g}_{j,0} + h_1 \cdot \hat{g}_{j,1} + h_2 \cdot \hat{g}_{j,2} + h_4 \cdot \hat{g}_{j,4}$$

In the created coding vector $\tilde{\boldsymbol{g}}$, each index for which an pivot element has been identified in $\hat{\boldsymbol{G}}^{\mathsf{T}}$ is equal to the index in $\boldsymbol{h}$. However for indices where no pivot element has been identified in $\hat{\boldsymbol{G}}^{\mathsf{T}}$, the result is a sum of products. Thus g' of the indices in $\tilde{\boldsymbol{g}}$ can be easily specified, whereas the remaining $g - g'$ cannot.

D. Drawing Recoding Vectors and Linear Dependency

The simplest approach to insure that all $g - g'$ elements are in A_2 is to draw all elements in $\boldsymbol{h}$ randomly from A_1. If one of the resulting elements in $\tilde{\boldsymbol{g}}$ is in A_1 the embedded information is incorrect, $\boldsymbol{h}$ is discarded, and a new $\boldsymbol{h}$ generated. The elements $\hat{g}$ are in q and thus if g' is not very low, we can assume that the resulting $g - g'$ σ's are uniformly distributed. The probability that one resulting index is in A_2 is $\frac{|A_2|}{q}$ and thus the probability that all $g - g'$ are not in A_2 is.

$$P_{\text{discard}} = 1 - \left(\frac{|A_2|}{q} \right)^{g - g'} \tag{3}$$

The probability P_{discard} is highest when g' is low. In particular the mean P_{discard} is of interest, this is system dependent as it dependent on when recoding is performed. Specifically the distribution of g' must be known to calculate the mean P_{discard} exact. However, recoding will most likely not be performed when g' is very low as this would mean that the recoding node hold little information. Thus we assume that $g' \geq g/2$. However, when $|A_2|$ is increased, $|A_1|$ is decreased which in particular impacts coding at a source.

Additionally consider the case where a node has only received trivial coded packets. In this case all valid $\boldsymbol{h}$'s are 0 for the indices where the node has received no symbols, and thus $\tilde{\boldsymbol{g}}$ will also be zero for these indices. Thus $0 \notin A_1$, and we have established that $0 \notin (A_1 \cup A_2)$ is required.

When a coded symbol is received there is a non-zero probability that the symbol is a linear combination of the already received symbols. A source that knows nothing about the symbols of a receiver, must code packets at random and hope that the sent symbol is useful at the receiver. In this case, the probability that a symbol is linear dependent at a receiving node is a function of the field size, q, and the rank deficiency at the receiving node, $g - g'$ [7]. As a source has pivot elements for all symbols, the usable field size is reduced from q to $|A_1|$.

$$P_{\text{dependent}} = \frac{1}{|A_1|^{g - g'}} \tag{4}$$

In particular the mean $P_{\text{dependent}}$ is of interest. Again the distribution of g' is necessary to calculate this value exact. However, if $A_1 \geq 2$, 2 is also the minimal field size, and the generation size g is not very low, this overhead is small [8]. If the sender does not hold all symbols uncoded, $\text{rank}(\hat{\boldsymbol{G}}^{\mathsf{T}}) < g$ this probability is less straight forward and depends on the correlation of the symbols at the sender and receiver.

Both Equation (4) and (3) should be low and hence there is a trade-off between $|A_1|$ and $|A_2|$. Ff q is not limited both probabilities can be arbitrary small. If $q = 2^{32}$ it is possible to choose $|A_1|$ and $|A_2|$, such that both mean P_{discard} and $P_{\text{dependent}}$ are extremely small regardless of g, and thus we can neglect them. If $q = 2^8$, $g = 128$, and we choose $|A_1| = 2$ then P_{discard} for $g' = g/2 = 64$ is 53%. If we assume that g' is uniformly distributed between $g/2$ and g the mean P_{discard} is 30%. See [9] for a small script to calculate these probabilities. Notice that recoding should first performed only be performed on the coding vector, and subsequently on the symbol only if the coding vector is usable. In this way the computational overhead from generating an unusable coded packet is very small.

III. System Example & Performance

To illustrate how intrinsic information in the coding vector could be used in a real system, we consider the following wireless cooperative scenario. N nearby sinks want to download the same data from some service provider. Each sink has a cellular link and a local wireless link. With the cellular link the node is connected to a Base Station (BS) that provides access to the service. We assume that a systematic random approach is used at the BS to reduce the computational overhead [8]. All sinks are interconnected via the local wireless link. In order to improve the download speed, conserve cellular bandwidth, conserve energy, etc. the sinks cooperate on downloading the data [10].

If the cellular links are orthogonal the BS should split the content into N parts and transmit each part to one sink. As each sink receives unique symbols from the BS these symbols should be forwarded to the rest of the sinks in the cluster. As the BS is unicasting data to each sink, it is straightforward to ensure that the cluster combined receives all the symbols.

If the cellular links are non-orthogonal, the sinks cannot know if they hold unique symbols without signalling, as other sinks could also have received the symbols. Thus when a symbol is received from the BS it is only stored. In this case the BS is broadcasting the data to the cluster and therefore it is more complicated to ensure that each symbol has been received by at least one sink in the cluster. There exists several approaches, and we assume that such an approach is used.

In both cases erasures on the broadcast link causes each sink to hold a subset of the original symbols, and potentially some coded symbols. Furthermore we assume that each symbol has been received by at least one sink in the cluster. Hence the cluster can cooperate locally and exchange symbols until all sinks can decode the data. In this local repair phase the introduced intrinsic idea can be used to reduce the computational complexity.

Initially each sink knows nothing about what the other sinks hold, as it has not received any symbols from them. If a sink has no information about the other sinks, it is necessary to fall back to the traditional RLNC approach. However, as intrinsic information is embedded into the coding vector, the sink simultaneously communicates what it has and does not have. Thus a sink starts to code more intelligently when it has received one coding vector from each of the other sinks in the cluster.

Based on the knowledge of what the other sinks need, each sink can create coded symbols that are useful for all other sinks. The coding sink identifies, for each of the other sinks, a symbol that is needed by that sink and for which the coding sink has a pivot element. In the worst case the coding sink must choose a different symbol for each sink. In the best case they all need the same symbol and the coding sink can simply send that symbol. Whenever a sink receives a symbol from another sink, it updates its local state information about the sending node. To keep this information up to data the nodes can transmit in round-robin fashion.

A. Computational Complexity

Here we consider the computational complexity as the number of row operations performed, where each row operation is either multiply and add or multiply and subtract. As we consider a binary extension field addition and subtraction is identical.

Two things influence the computational complexity in this system. One is the amount of coding needed to recode a symbol at a sink, or alternatively how many symbols held at the sinks that are combined. The other is the density of the resulting coding vector, as this indicates the amount of work necessary to decode the symbols at the receiver. The density is the ratio of non-zero elements in a coding vector, and can be calculated with Equation (5), for a coding vector $\boldsymbol{h}$ with a generation size g.

$$D(\boldsymbol{h}) = \frac{\sum_{k=1}^{g}(h_k \neq 0)}{g} \tag{5}$$

When a sink codes i symbols together, the first symbol is multiplied with an element drawn from q and copied to a buffer. For each subsequent symbol an element is drawn from q and multiplied onto the symbol, the results is then added to the buffer.

In a traditional RLNC, all received symbols are combined every time, hence Equation (6). With the intrinsic approach the number of combined symbols is at most equal to the number of sinks, if a different symbol is coded for each sink. As one sink is sending, the number of receivers is $N-1$. In the best case a single symbol can simply be forwarded, if there is a symbol for which all receivers have no pivot element. This gives the bound in Equation (7) and (8).

$$R_{\text{RLNC}} = g' \tag{6}$$

$$R_{\text{intrinsic}} \leq \min(N-1, g') \tag{7}$$

$$R_{\text{intrinsic}} \geq 0 \tag{8}$$

When a sink decodes it identifies the first non-zero element in $\tilde{\boldsymbol{g}}$. The coding vector and the coded symbol is then multiplied with this elements inverse to obtain a pivot element in the coding vector. If the sink holds another coding vector that has pivot element for the same index it is then subtracts, the coding vector and coded symbol from the received coding vector and symbol. This substitution is continued until the data is decoded. Thus the computation complexity of coding and decoding are comparable. For the traditional RLNC approach all coded symbols are completely dense. For the intrinsic approach the density is at worst equal to the sum of the rank deficiency at the coding sink, and the number of symbols that are coded together. The reason is that for all indices' where no pivot element has been identified, the result is a non-zero index in $\tilde{\boldsymbol{g}}$. The minimum density is similar but only a single symbol is forwarded.

$$D_{\text{RLNC}} = g \tag{9}$$

$$D_{\text{intrinsic}} \leq \min(N-1+g-g', g) \tag{10}$$

$$D_{\text{intrinsic}} \geq \min(g-g', g) \tag{11}$$

B. Results

Thus we know the computational requirements for coding a symbol, and decoding the symbol as a function of g'. Decoding is completed when $g' = g$ and thus we can calculate the number of operations needed to go from g' to g, which is done by calculating the survival function, or one minus the cumulative distribution function. In particular the survival function here specifies the number of expected remaining operations that must be performed from an particular g' until decoding is completed. The normalized survival function is plotted for $g = 128$ and $n = 4$ on Figure 2. On the x-axis is g' which indicates the starting point of the nodes g'. If this number is divided by g it can be interpreted as the Packet Error Probability (PEP) for the broadcast channel, e.g. if $g' = g/2$ then half of the g symbols was lost and thus PEP=.5. Additionally g' equal to g and 0 represent the extreme cases of PEP=0% and 100% respectively. The latter case also represent the case when a non-systematic code is used, as no trivially coded packets are received. On the y-axis is the survival function of the number of operations. Note that for $C_{intrinsic}$ and $D_{intrinsic}$ the area between the upper and lower bound is filled.

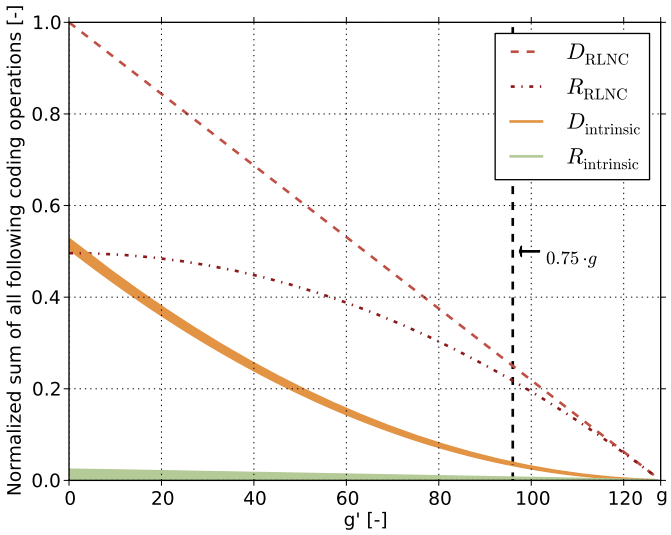

Fig. 2: Number of operations needed to finish decoding as a function of g'.

Figure 2 shows how the amount of coding operations needed for recode and decode is significantly reduced, when information is conveyed with the intrinsic approach. We have marked the case where $g' = .75 \cdot g$, which corresponds to a PEP of 0.25. For this case the coding complexity is decreased by 96 %, and decoding complexity is decreased by 82 %.

If g' is lower the numeric reduction in complexity is higher. However the reduction in percent is lower. Unless $g' \approx g$ the reduction in amount of coding is very significant, and as a result we expect the computational load to be decreased considerably for both recoding and decoding nodes.

IV. Conclusion

We have proposed the idea of embedding information into the coding vector that accompanies a coded symbol in a random linear network coding system. We have also introduced an approach for how this information can be embedded, and retrieved in a practical system. To evaluate the idea we have outlined a simple suggestion for what information could be conveyed and how the system could operate in a simple cooperative scenario. The results demonstrate that the amount of coding performed can be significantly reduced when the conveyed information is used during recoding of symbols. Additionally the density of the coding vectors is reduced which makes decoding at the receiver less computational demanding.

The outlined system is meant to demonstrate the idea, and how it could be implemented in a practical system. However, additional work is necessary to produce a complete system and a working protocol. Furthermore such a system should be evaluated in a realistic scenario, e.g. simulated with proper channel models.

Acknowledgment

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

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] W. Feller, *An Introduction to Probability Theory and Its Applications, Volume 1*. Wiley, 1968.

[3] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "Xors in the air: practical wireless network coding," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '06)*. ACM Press, September, 11-15 2006, pp. 243–254.

[4] T. Ho, R. Koetter, M. Medard, 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

[5] B. Haeupler, "Analyzing network coding gossip made easy," *CoRR*, vol. abs/1010.0558, 2010.

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

[7] A. Eryilmaz, A. Ozdaglar, and M. Medard, "On delay performance gains from network coding," *Information Sciences and Systems, 2006 40th Annual Conference on*, pp. 864–870, March 2006.

[8] 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.

[9] M. V. Pedersen, J. Heide, and F. H. Fitzek. (2011, Jun.) The cone project homepage. [Online]. Available: http://mobdevtrac.es.aau.dk/cone/wiki/intrinsic

[10] F. Fitzek and M. Katz, Eds., *Cooperation in Wireless Networks: Principles and Applications – Real Egoistic Behavior is to Cooperate!*, ser. ISBN 1-4020-4710-X. Springer, April 2006.