Low Computational Complexity Network Coding For Mobile Networks

Heide, Janus

Publication date:
2012

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Heide, J. (2012). *Low Computational Complexity Network Coding For Mobile Networks*.

Low Computational Complexity Network Coding for Mobile Networks

Ph.D. Thesis

Janus Heide

AALBORG UNIVERSITY
DENMARK

August 2012

Department of Electronic Systems
Frederik Bajers Vej 7
DK-9220 Aalborg Ø
Denmark

Title
Low Computational Complexity Network Coding for Mobile Networks

Date of submission
9th of August 2012

ISBN
978-87-92328-87-8

Academic advisors
Professor Frank H.P. Fitzek, Aalborg University, Denmark
Professor Torben Larsen, Aalborg University, Denmark

Internal examiner
Associated Professor Jan Østergaard, Aalborg University, Denmark

External examiners
Professor Mario Gerla, University of California, Los Angeles
Professor Wolfgang Utschik, Technical University of Munich, Germany

Abstract

Network Coding (NC) is a technique that can provide benefits in many types of networks, some examples from wireless networks are: In relay networks, either the physical or the data link layer, to reduce the number of transmissions. In reliable multicast, to reduce the amount of signaling and enable cooperation among receivers. In meshed networks, to simplify routing schemes and to increase robustness toward node failures.

This thesis deals with implementation issues of one NC technique namely Random Linear Network Coding (RLNC) which can be described as a highly decentralized non-deterministic intra-flow coding technique. One of the key challenges of this technique is its inherent computational complexity which can lead to high computational load and energy consumption in particular on the mobile platforms that are the target platform in this work.

To increase the coding throughput several simplifications have been considered, such as decreasing field size, density, and coding systematically. In order to increase the benefits of these simplifications we considered different decoding algorithms. From a practical point of view we identified the conflict between recoding and a compact coding vector representation. This problem is relevant as the total overhead due to RLNC stems from the probability of linear dependent symbols but also from including the coding vector of the transmitted coded symbol. Finally, we present an approach that mitigates these problems and enable a significantly higher coding throughput. The approach is based on random but non-uniform combination of symbols in a generation. Unlike approaches that adapts the ideas from Fountain codes the presented proposal is not based on a degree distribution. The efforts and experience stemming from this work have been incorporated in the Kodo library and will be available for researchers and students in the future.

Chapter 1 introduces motivating examples and the state of art when this work commenced. In Chapter 2 selected publications are presented and how their content is related. Chapter 3 presents the main outcome of the work and briefly new important progresses in the state of the art. The final conclusions are drawn in Chapter 4.

Resumé

Network Coding (NC) er en teknik der kan give fordele i mange typer netværk, for trådløs netværk kan følgende eksempler nævnes: I *relay networks* enten på det fysiske eller datalink laget for at reducere antallet af transmissioner. I *reliable multicast* for at reducere mængden af signalering og muliggøre samarbejde mellem modtagere. I *meshed networks* for at forsimple *routing* systemet og for at højne robustheden mod driftsvigt af knuder.

Denne afhandling omhandler implementeringsudfordringer for en specifik NC teknik kaldet Random Linear Network Coding (RLNC) der kan beskrives som en decentraliseret ikke deterministisk *intra-flow* kodningsteknik. En af de primære udfordringer ved brug af denne teknik er dens beregningsmæssige kompleksitet, der kan resultere i en højt beregningsmæssige byrde og et højt energiforbrug, særligt på mobile platforme som dette arbejde er tilsigtet.

For at forøge kodningshastigheden er flere forsimplinger blevet overvejet f.eks. at reducere feltstørrelsen, reducere tætheden og at kode systematisk. For at øge fordelen af disse forsimplinger overvejede vi forskellige dekodningsalgoritmer. Fra et praktisk synspunkt identificerede vi konflikten mellem rekodning og en kompakt kodningsvektorrepræsentation. Dette problem er relevant da det totale overhead p.g.a. RLNC stammer fra sandsynligheden for lineært afhængige symboler, men også fra den inkluderede kodningsvektor. Endelig præsenterer vi en fremgangsmåde der reducerer disse problemer og muliggører en betydelig højere kodningshastighed. Fremgangsmåden er baseret på en tilfældig, men ikke uniform kombinering af symboler in en generation. I modsætning til fremgangsmåder der tilpasser ideer fra Fountain koder, er den præsenterede idé ikke baseret på *degree* distributioner. Indsatsen og erfaringerne fra dette arbejde er blevet inkorporeret i Kodo software biblioteket og vil være tilgængelige for forskere og studerende i fremtiden.

Kapitel 1 introducerer motiverende eksempler og den nyeste viden indenfor området da dette arbejde blev påbegyndte. I Kapitel 2 præsenteres udvalgte publikationer og hvordan deres indhold er relateret. Kapitel 3 præsenterer de vigtigste resultater af arbejdet og kort vigtige nye udviklinger i videnen på området. Den endelige konklusion drages i Kapitel 4.

Preface

This Ph.D. thesis presents the results obtained during my three years as a Ph.D. student in the Antennas, Propagation and radio Networking (APNet) section, Department of Electronic Systems, Aalborg University. This work was financed by the COoperation and NEtwork coding (CONE) project (Grant No. 09-066549/FTP) granted by the Danish Ministry of Science, Technology and Innovation. The thesis includes five selected papers out of my complete list of 39 publications.

Acknowledgments

I would like to thank the following for their help, support and cooperation during my time at Aalborg University. My supervisor Professor Frank H.P. Fitzek for; opening many doors, interesting ideas, strange ideas, and for making research fun. Morten V. Pedersen for the close cooperation during the last five years, it has been a great time. Kirsten Nielsen for helping with countless administrative challenges and for always going the extra mile. Professor Torben Larsen for his role in making this project materialize.

A special thanks to Professor Petar Popovski who first gave me the possibility to work in the interesting area of communication, and Rasmus Krigslund and Jesper Sørensen with whom I worked on many projects from the very first semester at Aalborg University.

Professor Muriel Médard for hosting me at MIT and priceless input in form of ideas and helpful ways of explaining unexplainable concepts. Qi Zhang, Peter Vingelmann, and Marcos Katz for interesting discussions and tireless improvements of manuscripts and notation. All the guys in the group, Stephan Rein, Achuthan Paramanathan, Martin Hundebøll and Peyman Pahlavani, who have kept me company for the last year. All my many colleagues in the APNet section.

Finally, to my family and friends for your support. I look forward to be able to spend more time with all of you. In particular thanks to Heidi.

The work included in this thesis is dedicated to my family.

Everything should be made as simple as possible,
but not simpler.

-Albert Einstein

Acronyms

APNet	Antennas, Propagation and radio Networking	vii
BS	Base Station	3
C2C	Car to Car	7
CONE	COoperation and NEtwork coding	26
CPU	Central Processing Unit	10
ENOC	Evolved Network COding	26
FEC	Forward Error Correction	24
FF	Finite Field	9
GPU	Graphical Processing Unit	10
HPC	High Performance Computing	25
MANET	Mobile Ad hoc NETwork	7
MBMS	Multimedia Broadcast Multicast Service	16
NC	Network Coding	1
NOCE	Network COding Evolved	26
OSI	Open Systems Interconnect	2
PC	Personal Computer	6
P2P	Peer to Peer	6
RLNC	Random Linear Network Coding	8
UDP	User Datagram Protocol	4
VANET	Vehicular Ad-hoc NETwork	7
WLAN	Wireless Local Area Network	16
TCP	Transmission Control Protocol	4

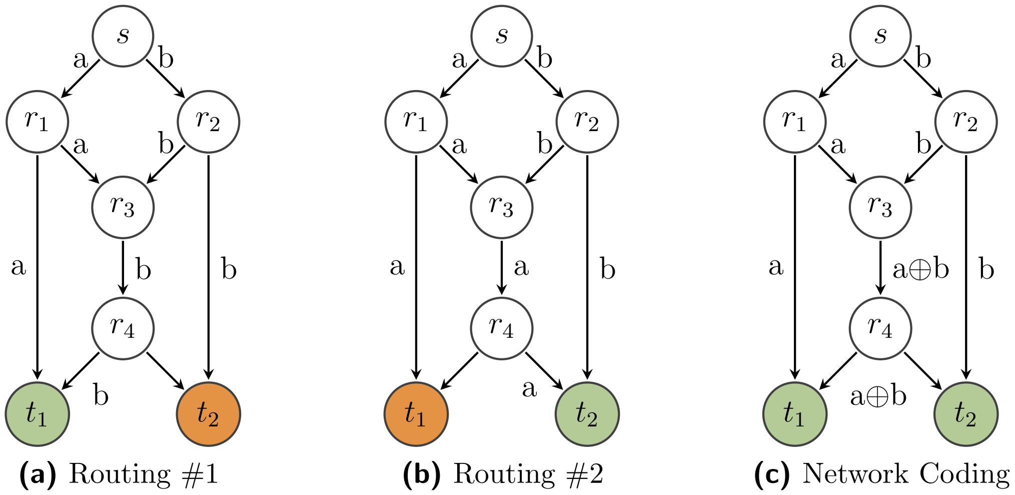
Contents

Chapter 1

Introduction

1.1 Breaking with Store-and-forward Networks

Network Coding (NC) [40, 41] is a technology that have been demonstrated to provide benefits in many scenarios in both wireless and wire line networks. Perhaps more importantly it has changed the understanding of data distribution in computer networks. A famous NC example called *the butterfly*, elegantly demonstrates how NC can be used to solve a problem where the existing *store-and-forward* or *routing* approach is not optimal.



(a) Routing #1 **(b)** Routing #2 **(c)** Network Coding

Figure 1.1: The butterfly network with and without NC. A source s holds packets, a and b, both of which sinks t_1 and t_2 want to retrieve. All links in the example has unit capacity. Using routing one of the sinks can receive both packets, colored green, while the other sink only receive one, colored orange. With NC r_3 can combine two packets and both sinks can retrieve both packets as (a⊕b)⊕a = b and (a⊕b)⊕b = a.

The importance of this example network becomes clear when the *Max-Flow Min-Cut* theorem is applied. This theorem has stood as a cornerstone in information theory for half a decade [42] and fundamentally states that the maximum flow in such a network is defined by its bottleneck. For the butterfly network the min-cut from the source to each of the sinks is two hence, we would expect that both sinks can receive both packets. However, until NC was introduced no solution for this network had been found. NC breaks with the basic premise in *store-and-forward* networks where packets are treated as atomic units that can only be forwarded or discarded. Instead, flows of information can be mixed at any location in the network. Since its introduction, NC has been suggested as a means to improve performance on all layers in the Open Systems Interconnect (OSI) model [43–48] and some ideas have already been tested by researchers [44, 49–52].

Coding in communication systems is not new. Information to be sent from a source to a sink is represented in a way that is suitable for transportation over a network. In most cases the information is compressed in order to reduce the resources necessary to transport it, denoted *source coding*. As the data is sent from one node to another it traverses the communication channel that is established between the nodes, denoted a link. When the data is transmitted over such a link errors can occur. In order to detect and correct these errors another type of coding, *channel coding*, is employed. Network coding adds a layer of coding in between these two existing layers. This space is partly occupied by *Fountain codes* [53], noticeable LT and Raptor codes [54, 55].

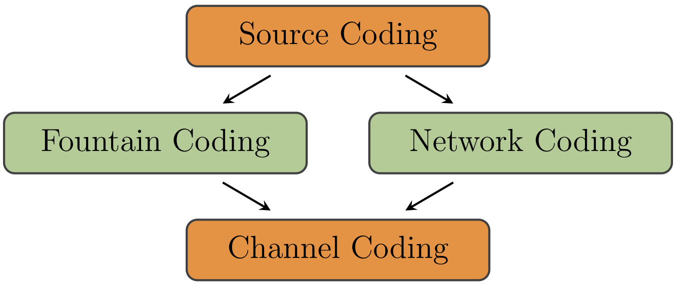


Figure 1.2: The hierarchy of coding, with NC inserted alongside Fountain Coding.

Intriguing theoretical properties and potential gains are reasons to consider NC. But the ideas must also be implemented in practical system which is the challenge faced by engineers. Ultimately the benefit of deploying NC must outweigh its cost, both from a technical and business point of view. But before considering such challenges we introduce some motivating examples that show why NC could become an important technology in future networks.

1.2 Device Centric Mobile Networks and Network Coding

The way users access information and use computers have changed significantly over the last couple of decades. The Internet has moved from a research project to forming one of the worlds most important pieces of infrastructure. In the last decade *social services* where users share content publically or privately have become increasingly popular. Also streaming of video and audio have become much more widespread. Traditional asynchronous one-to-one services such as email and web pages are still popular and important, but constitute a decreasing percentage of the total traffic [56]. The devices in use have also changed from desktops literally tethered to the wall, towards portable laptops and the highly mobile smartphones which are becoming the dominant platforms. In particular the rise of the smartphone has allowed users to access information anywhere and anytime.

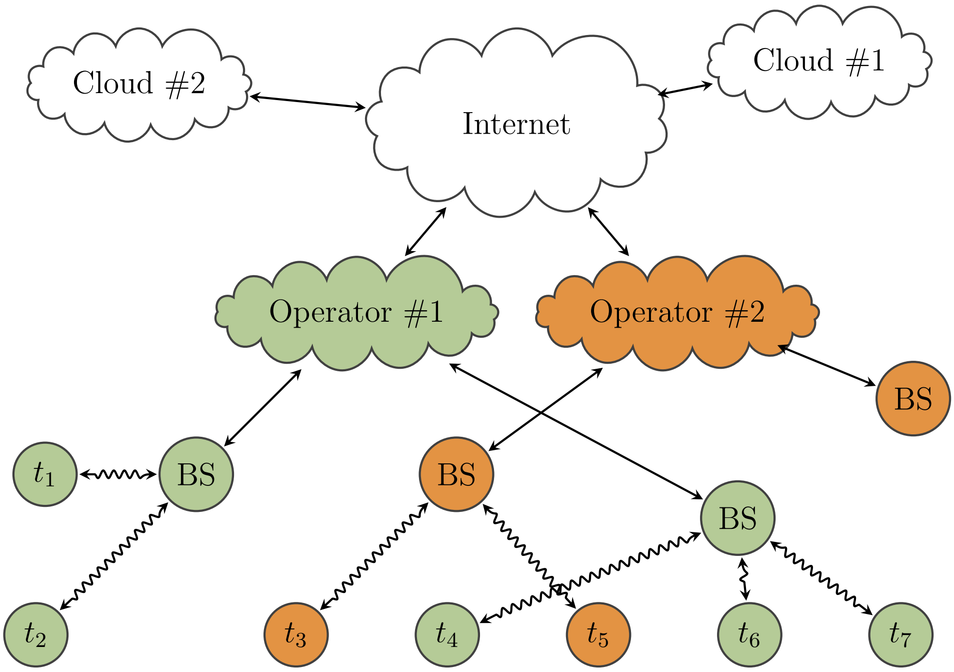


Figure 1.3: An overview of the network architecture that supports current *cloud services* that users can access via mobile phones and similar.

The cellular networks that fuel these devices have become faster and coverage has improved. However, they are still based on the very same premise as previous cellular networks, this is, users communicate with a Base Station (BS) that provide access to the Internet as illustrated in Figure 1.3.

This approach can be considered as a direct adoption of the topology form wire line phone networks, where the users are connected via a wire to a central station. This fits nicely with use cases where the data needed by the node is located on some centralized entity, a server. However, it does not permit nodes to communicate directly. Furthermore, the nature of wireless network offers possibilities that are currently not exploited since broadcasting is not supported neither at a BS nor from a user. In wireless networks the medium is shared between the users who transmit signals which propagate through the surrounding environment and can be overheard by others than the intended receiver. Broadcast could in some scenarios be utilized to make up for some of the challenges that wireless networks posses such as lower reliability and more dynamic link quality.

The architecture of the Internet is also predominantly centralized and highly regulated. In the *cloud*, information is stored at a single central authority as illustrated on Figure 1.3. Thus, if a user sends a message to another user right next to him/her, e.g. an email, the data would traverse the wireless link to the BS, the operators network, the Internet, to finally arrive at some central authority, for then to take a similar journey back to the other user. If the users could communicate directly the data could simply be send from one device to the other. In the case of text messages this will introduce a short delay but most likely not be noticed by the user. But if the user instead share a video, capacity will be needlessly wasted in the backbone network. This problem becomes even more apparent if one consider the case where a user want to send the video to multiple other users nearby. In this case the wasted bandwidth both at the source and in the backbone will increase proportionally with the number of sinks. If we consider the users t_3 and t_4 in Figure 1.3, they might be close in terms of location, but far apart from a network point of view. Furthermore, if the cloud is used to facilitate the data transfer, the network distance is likely to increase. This illustrates the fundamental trade-off between storage and bandwidth in computer systems. If you could store all of the Internet on your phone, you would not need an Internet connection, at least until your version got out of sync.

Fortunately the current systems have allowed for a multitude of services, many of which was not envisioned when the Internet was first developed and built. This shows the flexibility of the current solutions. The flexibility can partly be attributed to the simplicity of the Internet where almost all communication between end nodes is based on Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). On top of these two protocols a host of application layer protocols serving a vast range of purposes have been and is being built. The challenge is thus to efficiently support new uses without breaking or complicating the existing networks.

An example of such a network topology is cooperative networks [57]. In cooperative networks some of the burden of distributing content is moved from the source unto the the sinks. A simple example is that of reliable multicast in a local wireless network. Here a single source hold some data or stream that multiple sinks want to retrieve. Traditionally the sink can either unicast to each of sinks to ensure reliability or broadcast to everyone simultaneously and hope that the content is received correctly. With cooperation the sinks can exchange received data and hence increase the reliability of the distribution without increasing the burden of transmission at the source.

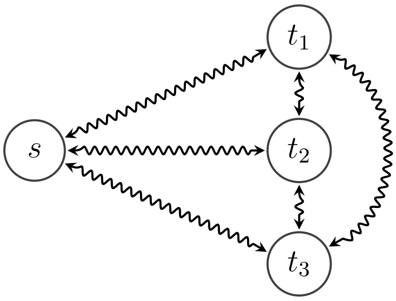


Figure 1.4: A simple cooperative network where the source s wishes to transmit some data reliably to the sinks t_1 through t_3. When the source broadcasts packets, the sinks will receive different subsets of the information due to packet erasures. Without cooperation the source must repair these erasures in order to ensure reliability. But if the sinks instead could cooperate and exchange packets that some of them are missing the load on the source can be reduce, also spectrum and energy usage at the sinks can be reduced. However, it is not trivial for the sinks to cooperate as they must determine who has lost which packets. With NC the requirement of obtaining this information can be removed which makes it much simpler to implement such a cooperative system.

In this scenario NC can be used to reduce the complexity of the cooperation. With traditional broadcast based cooperation the sinks would have to exchange information about what pieces of the data they have received in order for other sinks to be able to transmit missing pieces. If the number of nodes is low this is doable but if the number of sinks become high the amount of signaling will increase. Additionally, when sinks are helping each other out they will have to send data that some of the other sinks have already received, and thus their transmission will only be helpful for some of the other sinks. With NC the sinks can combine the data they have received and transmit the combination to other sinks without knowledge of what the other sinks have received. Thus, the signaling complexity is greatly reduced, and at the same time a sink can send data that is useful for more of the other sinks.

Distributing files or streaming data such as audio and video can be costly due to the upkeep of data centers and consumed outgoing bandwidth. To mitigate this problem a decentralized approach to content distribution have been in widespread use for some years. These systems are known as Peer to Peer (P2P) and allow a user to take part in the distribution of content, typically big files [58]. This concept have also been adopted for mobile devices, not surprisingly called mobile P2P [59].

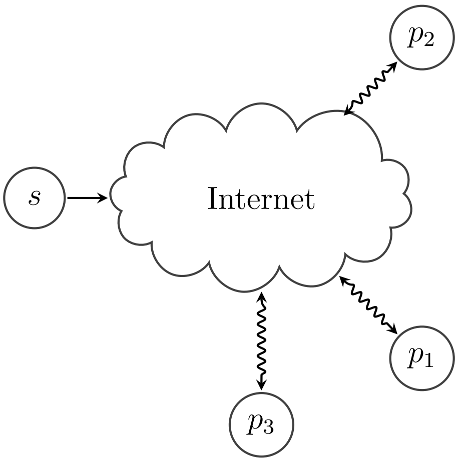


Figure 1.5: A P2P network where peers form an overlay distribution network in order to offload the user that originally introduced the content. The peers query other users for parts of the data they miss and in term respond to queries by others. In order to maintain a set of suitable source peers all peers periodically sends queries to new potential source peers.

Originally, P2Ps systems were intended for distribution of large files to Personal Computers (PCs) using a relatively stable Internet connection. For other use cases the ideas used in such systems might not be directly applicable. For example, in the case of P2P live video streaming between mobile phones. In this case there is a strict delay requirement, if information arrives to late it will be unusable for the sink. Additionally, the links of the sinks can be much more dynamic as the users are on wireless connection. Therefore, this use case is much more dynamic and the system must be able to adapt much faster.

With NC the solutions can become simpler, instead of requesting specific chunks of data the sinks simply ask for some symbols from a series of other sinks. Importantly, end-to-end codes cannot be used in this way, as they do not support recoding. Hence, a sink would have to decode the information completely before it can start contributing to other nodes which can introduce an unacceptable delay.

A final example where NC can be applied is meshed networks which can be used to create a wireless infrastructure to allow Internet access in a city, or formed between mobile sensors. To enable the nodes in the network to access the Internet some of the nodes will have a wireless connection and act as gateways through which the nodes traffic will be routed [60]. Similarly, for sensor networks where the task of the gateways are to collect measurement data and forward it via a global link, or alternatively store it until some external entity passes by and collects it. Mobile Ad hoc NETworks (MANETs) [61] are similar but has highly dynamic topologies that are often considered for Car to Car (C2C) communication called Vehicular Ad-hoc NETworks (VANETs) or military purposes such as connecting a large number of combat units, infantry, tanks, artillery, airplanes etc., on the battlefield in order to share information in real time. Such networks posses additional challenges in terms of quickly changing network typologies which introduces the need for robust routing schemes.

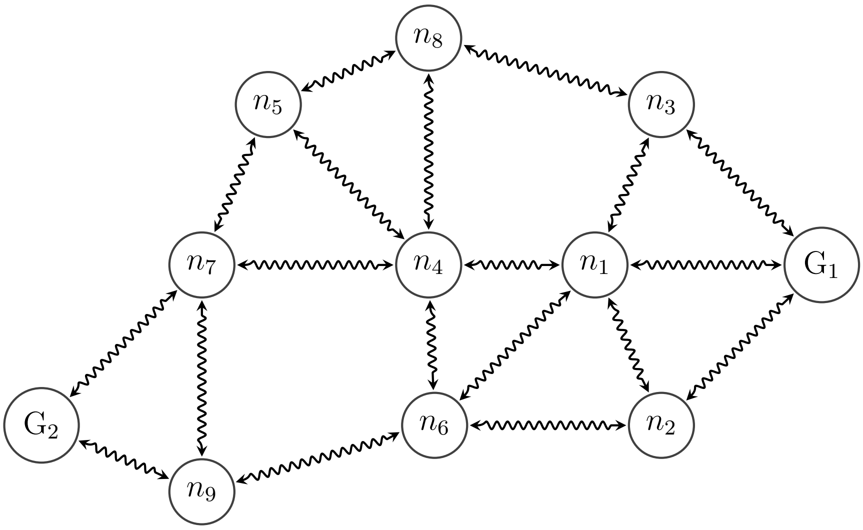


Figure 1.6: Multiple nodes generate, consume or both, data that must be forwarded to or received from a gateway. The topology of the network may change slowly or fast which means that the routes packets traverse to or from the gateway must sometimes be updated. If the nodes are battery driven the routing approach must avoid high load on single nodes as this may cause fast battery depletion and create dead spots in the network.

In such a system NC can be used to create simpler routing schemes and/or to efficiently spread the information in the network, so it does not become unavailable should a particular node stop operating. This makes it simpler to create robust multipath or corridor based routing protocols.

The provided examples are all less centralized and more loosely organized networks when compared to the cellular networks used today and the Internet in general. Thus, it seems logical that new ideas and approaches could be beneficial in these topologies and other networks that does not adhere to the strictly centralized architecture. Specifically if the networks become more localization aware and allow users to exploit co-location by cooperation, they could perform better in scenarios such as those described. This could also offload centralized servers and backbone networks which could reduce the cost of the service providers or allow users to create independent services. This sounds enthralling, unfortunately the consequence can be much more complex networks. If the role of authority is no longer well defined, it becomes difficult to guarantee things such as security, authentication, and reliability. This work does not consider all of these problems but instead focus on the challenges of efficient communication in computer networks in terms of spectrum and energy.

One approach is to base such systems on Random Linear Network Coding (RLNC) as it presents a completely decentralized approach to data distribution. With RLNC nodes can encode and recode data with only minimal information about the state of the rest of the system. Thus nodes can exchange data without introducing a high level of signaling. As a consequence RLNC can be used to implement cooperation in networks where the added overhead of non-coding cooperative systems would be prohibitively high. Additionally, the reduced amount of signaling makes it particular suitable for highly dynamic networks, which is a typical property of mobile networks. Unfortunately RLNC is computational demanding which can be a significant challenge when it is utilized in practical systems.

For successful deployment on mobile devices RLNC must be adapted to preserve computational and energy resources, as the nodes in these networks traditionally have low computational capabilities and are battery driven. The good news is that the computational capabilities of mobile phones have increased rapidly during the last years which can be observed from the increasing highest clock frequency available which was 200 MHz in 2005, 600 MHz in 2007, 1 GHz in 2009, 1 GHz dual-core in 2011, and currently 1.5 quad-core in 2012 [62–67]. This is an increase of 30 times, where as the battery capacity has only doubled during the same time. This shows that today and in the future the problem of energy is becoming more important. However utilizing this computational power still comes with a cost in terms of energy and therefore it is still relevant to aim for low computational demanding solutions.

1.3 Random Linear Network Coding for Practical Systems

In 2009 when this project started RLNC had been proposed as an theoretical efficient approach of performing NC [68]. Coding is performed over a Finite Field (FF) which makes the code linear [69] as all valid symbols can be combined to form new valid symbols. The approach is random and encoding and recoding can be performed completely decentralized at nodes in the network. The basic approach in RLNC is to divide the original data into k symbols and combine these chunks at random. Each symbol is multiplied with a scalar drawn at random from a FF, and all the results are added to obtain a coded symbol of the same size as the original symbols. In order for a node to decode the original data it must collect slightly more independent coded symbols as there are original symbols $(1+\epsilon)k$, where ϵ is the overhead per symbol.

This slight overhead arises as the scalars are drawn at random and hence there is a non-zero probability that a received symbol is linear dependent on already received symbols. The overhead can be made arbitrarily small provided that the used FF is sufficiently large. To make RLNC practical for any size of data it was suggested to divide the original symbols into *generations* $\boldsymbol{M}$ [70]. This increases the overhead as enough symbols must be collected for each generation and can add additional signaling [71].

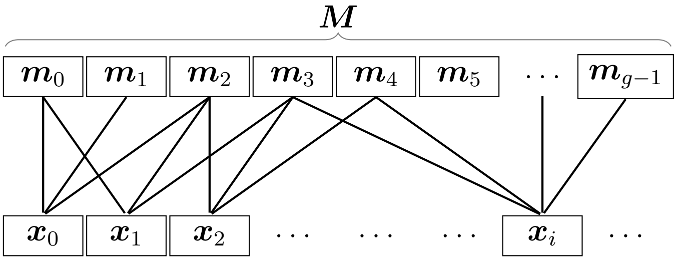


Figure 1.7: Coded symbols are created from the original data $\boldsymbol{M}$ that is divided into symbols, denoted $\boldsymbol{m}_i$. Each coded symbol $\boldsymbol{x}_i$ is a combination of some or all of these original symbols. The benefit over non-coding approaches is that the original data can be recovered from any sufficiently large set of coded symbols. Without coding, all unique original symbols are needed in order to recover the original data [72].

To obtain a low overhead, coding has been performed at random over a large field typically a binary extension field of size 2^8 or 2^{16}. Under such settings it is not trivial to obtain an acceptable level of coding throughput which was also demonstrated by the first attempt [73]. The problem was especially apparent when RLNC was first deployed on mobile phones of the time [6]. Thus, the complexity of RLNC had been mentioned as a challenge with a significant impact on the coding throughput. However, reducing this complexity had not attracted much attention and all presented solutions where based on an approach to RLNC that was proposed in a theoretical context [68, 74].

Nevertheless, several P2P like systems for large scale content distribution had been implemented and tested. In the similar [75, 76] the file distribution system Avalanche was presented, however very little information about the throughput of the coding was conveyed. In [51] which presents a P2P system for live video streaming this problem was thoroughly presented. In order to achieve an acceptable throughput the authors used low generation sizes and very large blocks, in practice this reduces the usefulness of the coding considerably. None of these considered mobile platforms and to reduce the computational load they operated RLNC at settings where the coding complexity became less important. However, doing so may incur to high cost in terms of overhead for some use cases and may not be feasible on some platforms with low computational capabilities.

In order to increase the coding throughput some efforts had focused on utilizing Graphical Processing Units (GPUs) or parallelism [77, 78]. It had been demonstrated that both approaches could significantly increase the coding throughput when the powerful peaces of hardware were commandeered to perform the coding. However, these efforts targeted PCs and had little prospect of ever benefiting mobile devices. At the time multiple core Central Processing Units (CPUs) for mobile devices was still in the horizon and even though phones with a GPU had become available some years prior [63, 64], there was no easy way to utilize this processing power for general purposes, especially not if multiple platforms and devices are to be supported.

Particularly a high decoding throughput is important as in several proposed use cases the distribution is one-to-many and hence more nodes are decoding compared to encoding. Additionally the decoding node is often slower than the encoding node, e.g. in the case where a video is streamed from a server to some users on phones or laptops. Finally, the decoding nodes are typically also performing other tasks, e.g. playing back video.

As mobile devices of the time had much more modest computationally capabilities compared to PCs the challenge were even bigger on such devices. Additionally, on battery driven devices there were the consideration of energy consumption. Generally, when the utilization of the nodes processing capabilities increases so does the energy consumed. As a consequence, we identified coding complexity and practical solutions as areas where improvements were possible and necessary.

1.4 Thesis Motivation and Aims

The work in this thesis were motivated by making RLNC practical, with mobile devices as the platform of particular interest. Illustrating the practical feasibility of RLNC would be an important step towards its utilization in practical systems, and by demonstrating this on a platform with low computational capabilities other powerful platforms would automatically be included. By unlocking the power of RLNC, it would become available as a tool for solving practical problems of implementing future services and applications, such as cooperative systems and localization aware services.

The main goals of this thesis can be summarized as follows:

- Propose modifications to RLNC to reduce the coding complexity and increase the coding throughput.
- Propose, implement and test algorithms for promising low complexity variants of RLNC.
- Identify practical system problems when deploying RLNC and identify suitable system settings when used on mobile devices.

Chapter 2

Selected Contributions

The selected contributions in this section are all part of the effort to reduce the coding complexity of RLNC in order to increase the coding throughput and ensure that it can be used in practical systems. Figure 2.1 provides an overview of the selected publications and how they relate in order to communicate the chronology of the work in this thesis.

The work done in [25] has formed the foundation of the remaining papers. In this paper we considered a low complexity variant of RLNC by using the binary field and coding systematically.

To reduce the decoding complexity further in [15] we proposed to utilize the coding vector to communicate information about the state of the decoding matrix. This information was used to encode packets that are much less computational demanding to recode.

In [19] we investigated the reduction in decoding complexity when sparsely coded symbols are received. We considered modifications to the decoding algorithms to increase the coding throughput and observed the problem of fill-in.

Motivated by our investigation of sparse RLNC we performed a more thorough analysis of the coding overhead due to linear dependency in [17]. An important insight was that the overhead due to the representation of the coding vector contributes significantly in cases where recoded are used.

Finally, in [1] based on the insight from our previous work we proposed a new approach to perform RLNC. The proposed approach is binary, sparse and non-uniform, and it mitigates the problem of fill-in during decoding as well as the overhead due to the coding vector.

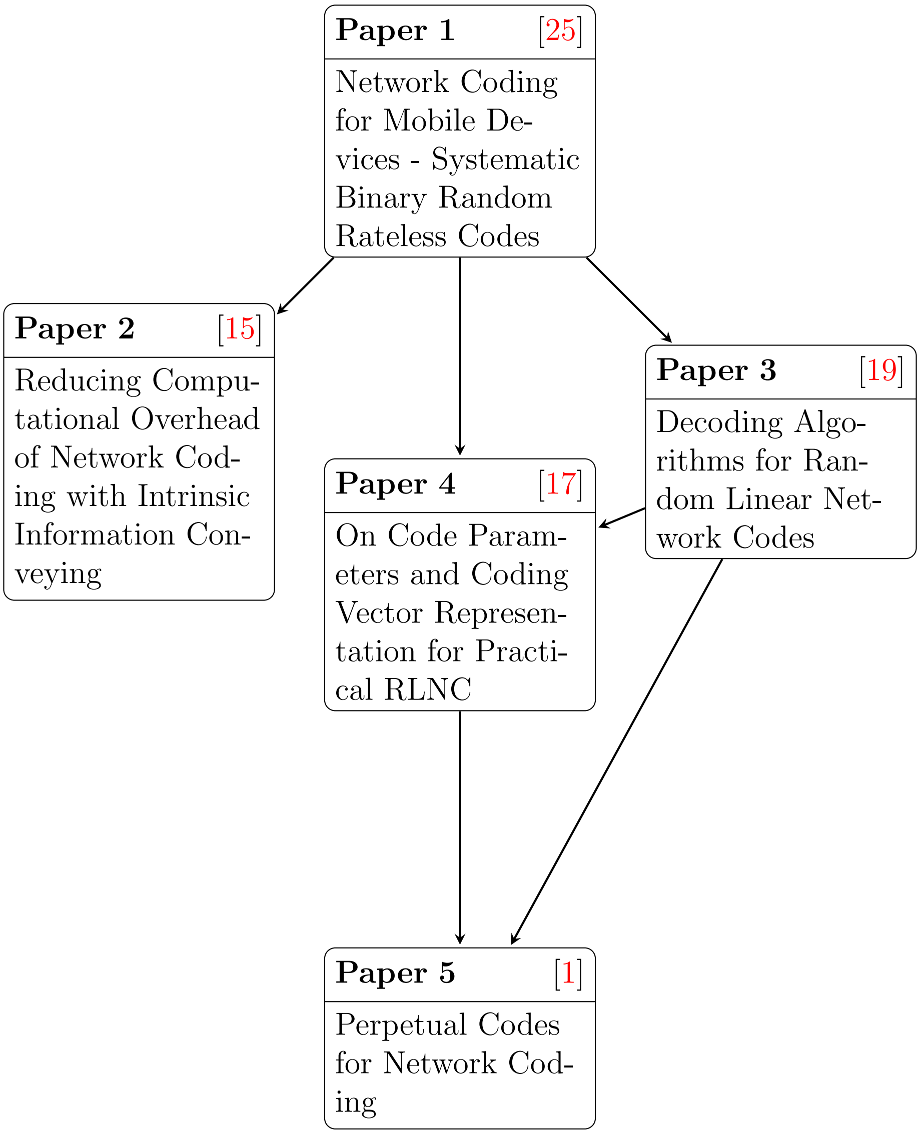

Paper 1 [25]
Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes
Paper 2 [15]
Reducing Computational Overhead of Network Coding with Intrinsic Information Conveying
Paper 3 [19]
Decoding Algorithms for Random Linear Network Codes
Paper 4 [17]
On Code Parameters and Coding Vector Representation for Practical RLNC
Paper 5 [1]
Perpetual Codes for Network Coding

Figure 2.1: The dependency between the selected contributions.

2.1 Paper 1

[25] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Torben Larsen. “Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes”. In: *IEEE International Conference on Communications (ICC) - Workshop on Cooperative Mobile Networks*. Dresden, Germany, June 14–18, 2009.

Motivation

The number of necessary transmissions in wireless reliable multicast scenarios can be dramatically reduced if the broadcast nature of the wireless medium is exploited. One technique that can be used to ensure reliability is RLNC, which can be used for both non-cooperative and cooperative systems. The control system of such a solution can be significantly simpler compared to solutions that do not employ coding, and at the same time the signaling overhead can be significantly reduced. However, the computational complexity added by RLNC is a drawback which must be addressed in particular when it is deployed on mobile devices such as mobile phones and tablets.

Content

This paper investigates how to perform reliable multicast from a single source using various methods for erasure correction. The overhead in terms of retransmissions from the source is analyzed for unicast, broadcast, and RLNC schemes all based on idealized feedback. For RLNC various generation and field sizes are considered along with both a non-systematic and systematic approach. The binary field and a systematic approach are considered in order to increase the coding throughput and thus decrease the energy consumption. The encoding and decoding algorithms used in the implementation are presented. The implementation is tested both on high end mobile device, a laptop, and on a low end mobile device, a mobile phone.

Main results

The overhead in terms of retransmissions from the source is evaluated for unicast, broadcast, and RLNC, for an increasing number of sinks. As the number of sinks grow the gain of using the RLNC approaches increases over broadcast. The performance of different field sizes and generation sizes is also compared, and the results demonstrate that it is unnecessary to use field sizes larger than 256 in the considered scenario. The encoding and decoding throughput are tested with the presented implementation for varying field sizes with and without systematic approach. Both the encoding and decoding throughput are approximately doubled for all tested settings and the coding throughput decreases as the generation increases.

Own Related Publications

In [5, 26] we presented an application that utilizes the approach presented in this paper to distribute pictures to a group of nearby users. In [8] we reported on link measurements in order to test the assumption of independent erasures at the users. In [13, 35] we considered a modified scenario where the BS transmits to the users over Multimedia Broadcast Multicast Service (MBMS) and the users cooperate using a local Wireless Local Area Network (WLAN).

[5] Morten V. Pedersen, Janus Heide, Frank H.P. Fitzek, and Torben Larsen. “A Mobile Application Prototype using Network Coding”. In: *European Transactions on Telecommunications (ETT)* 21.8 (8 2010), 738–749.

[8] Janus Heide, Péter Vingelmann, Morten V. Pedersen, Qi Zhang, and Frank H.P. Fitzek. “The Impact of Packet Loss Behavior in 802.11 b/g on the Cooperation Gain in Reliable Multicast”. In: *IEEE Vehicular Technology Conference (VTC) - Wireless networks Symposium.* accepted. Québec, Canada, Sept. 3–6, 2012.

[13] Qi Zhang, Janus Heide, Morten V. Pedersen, and Frank H.P. Fitzek. “User Cooperation with Network Coding for MBMS”. In: *IEEE GLOBal COMmunication conference, exhibition & industry forrum (GLOBECOM) - Communication Software, Services, and Multimedia Applications Symposium.* Houston, Texas, USA, Dec. 5–9, 2011.

[26] Morten V. Pedersen, Janus Heide, Frank H.P. Fitzek, and Torben Larsen. “PictureViewer - A Mobile Application using Network Coding”. In: *European Wireless Conference (EW).* Aalborg, Denmark, May 17–20, 2009.

[35] Qi Zhang, Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, Jorma Lilleberg, and Kari Rikkinen. “Network Coding and User Cooperation for Streaming and Download Services in LTE Networks”. In: *Network Coding: Fundamentals and Applications.* Ed. by Muriel Medard and Alex Sprintson. Academic Press, Oct. 17, 2011. Chap. 5, pp. 115–140.

2.2 Paper 2

[15] Janus Heide, Qi Zhang, Morten V. Pedersen, and Frank H.P. Fitzek. "Reducing Computational Overhead of Network Coding with Intrinsic Information Conveying". In: *IEEE Vehicular Technology Conference (VTC) - Transmission Technologies Track*. San Francisco, CA, USA, Sept. 5–8, 2011.

Motivation

The computational complexity at decoding sinks can be significantly lowered if the coded symbols received by the sink are sparse. Unfortunately this also increases the probability that such coded symbols are linearly dependent on previously received symbols, which increases the overhead in terms of retransmissions. However, if the encoding node knows a set of symbols to either include or exclude it can create sparse coded symbols with the same or higher probability of being linearly independent. However, conveying such information from one or more sink to the nodes that encode or recode entails an overhead in form of signaling which is unwanted.

Content

In this paper we proposed an approach where a FF was divided into sub fields where each field denotes some state information available at the coding node. Thus, this information is embedded into the coding vector whenever a node encodes or recodes a symbol and can be extracted at all receiving nodes before the symbol is decoded. We proposed to use this information for reducing the decoding complexity among a set of cooperative sinks that all perform recoding, by embedding information about the decoding matrix, and described an approach that enable this. This allow other nodes to create sparse coded symbols which requires less computational effort to decode.

Main results

In the considered cooperative scenario a group of nodes cooperatively download some content using their cellular links and cooperate by exchanging coded symbols over an orthogonal local network. At the evaluated settings with a generation size of 128 and four cooperating nodes the complexity of decoding was reduced with 82% when each of the nodes started with 75% of the content.

Own Related Publications

In [39] we described the core idea of embedding information into the coding vector and proposed a possible approach to implementing it.

[39] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Qi Zhang. “Intrinsic Information Conveying in Network Coding”. Pat. 800.0406.U1 (US), pending. Mar. 19, 2010.

2.3 Paper 3

[19] Janus Heide, Morten V. Pedersen, and Frank H.P. Fitzek. "Decoding Algorithms for Random Linear Network Codes". In: *IFIP International Conferences on Networking - Workshop on Network Coding Applications and Protocols (NC-Pro)*. Vol. 6827. Lecture Notes in Computer Science. Valencia, Spain, May 12, 2011, pp. 129–137.

Motivation

When decoding sparse coded symbols using standard RLNC decoding algorithms an effect sometimes called *fill-in* can be observed. Basically the sparse coded symbols are combined with other sparse symbols which together form less sparse symbols. As decoding progresses the symbols in the decoding matrix will become less sparse which means that the desired effect of reduced decoding complexity is lost. Therefore it is necessary to adapt the decoding algorithms in order to ensure a low computational load on the decoding node.

Content

In this paper we proposed, implemented and evaluated a series of simple optimizations to the *on-the-fly* version of Gaussian elimination that we use for decoding in our RLNC implementations. We tested each optimization on its own and also some of them combined in order to determine when they provided the biggest benefits.

Main results

The proposed optimizations were tested by measuring the number of row operations performed during decoding of a generation. A field size of two was used, and the generation sizes were varied from 16 to 512. Additionally, both traditional RLNC and a sparse variant were tested. In both cases the results showed that it was possible to reduce the operations performed on the coding vector to approximate half. For the dense case a reduction in row operations on the coded symbols of up to 10% was observed. For the sparse case a reduction of up to 20% in row operations on the coded symbols was observed.

Own Related Publications

In [20] we introduced the Kodo network coding library in which the algorithms were implemented.

[20] Morten V. Pedersen, Janus Heide, and Frank H.P. Fitzek. "Kodo: An Open and Research Oriented Network Coding Library". In: *IFIP International Conferences on Networking - Workshop on Network Coding Applications and Protocols (NC-Pro)*. Vol. 6827. Lecture Notes in Computer Science. Valencia, Spain, May 12, 2011, pp. 145–153.

2.4 Paper 4

[17] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Muriel Médard. “On Code Parameters and Coding Vector Representation for Practical RLNC”. In: *IEEE International Conference on Communications (ICC) - Communication Theory Symposium.* Kyoto, Japan, June 5–9, 2011.

Motivation

When analysis, simulation or practical system that is based on RLNC is presented, the generation size, field size and sometimes also the density are important parameters that describe how coding is performed. These parameters influence both the coding throughput and the overhead due to linear dependency, and should be chosen with care as improper values can result in a system that is practically unfeasible. Furthermore, we wish to direct attention towards the fact that using a pseudo random function to compress the coding vector makes recording impossible. Conversely, the overhead from the coding vector must be included when the total overhead is evaluated.

Content

In this paper we analyze the influence of the parameters generation size, field size, and density of the probability of linearly dependency. We explain why the commonly used assumption: that coding vectors can be represented with a seed if a pseudo random function is used at the coding nodes, is not applicable in cases where recoding is used. We introduce several new approaches to represent the coding vectors and present their respectively overhead. Finally we analyze the total overhead from both linear dependency and the coding vector representation. Based on this we discuss suitable choices of the parameters for different types of applications.

Main results

The provided analysis enables calculation of the overhead due to linear dependency from RLNC and sparse RLNC variants, as well as the overhead from the resulting coding vectors. The generation size should be specified based on the delay requirements of the considered application. Based on this the remaining parameters can be chosen in order to minimize the overhead. The results show that for generation sizes below 256 a field size of 2^8 provides the lowest total overhead, for generation sizes above 256 a field size of 2 provides the lowest total overhead.

2.5 Paper 5

[1] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Muriel Médard. "Perpetual Codes for Network Coding". In: *IEEE Transaction on Mobile Computing* (2012). submitted.

Motivation

Sparse variants of RLNC does not necessarily reduce the computationally of decoding. Therefore, we seek a new approach to perform the coding that is sparse, but allow decoding to be performed in a way where fill-in during decoding is avoided or reduced. Furthermore, the approach should not make it unpractical to combine coded symbols as this would prohibit recoding. Finally, it should be possible to represent the coding vectors in a compact way in order to reduce their contribution to the total overhead.

Content

In this paper a binary code is introduced where all symbols are not combined at random. Instead only neighboring symbols are combined at random. We describe how encoding, decoding and recoding can be performed, provide suitable algorithms for all operations, and propose optimizations to the coding procedures. We provide analytically upper and lower bounds on the coding complexity and overhead due to linear dependency. Finally, we provide access to our C++ implementation of the proposed approach. We report on measured coding throughputs obtained with the implementation and compare the results with our own similar implementation of traditional RLNC.

Main results

The results show that the proposed approach to performing RLNC can deliver an overhead due to linear dependency arbitrarily close to that of RLNC but at a significantly reduced coding complexity. The gain in the coding throughput grows with the generation size and is approximately one order of magnitude higher for the largest tested generation size of 2048. We also provide a new view on how recoding can be performed which is particular relevant for the proposed approach.

Chapter 3

Main Outcome

This chapter outlines the primary outcomes stemming from the work conducted during this Ph.D. study. The contribution has been centered around different efforts which are listed below.

1. Practical problems in mobile systems
2. Reduced coding complexity
3. The Kodo library
4. Steinwurf
5. The ENOC/NOCE project
6. The NBC demonstrator

The primary contributions to the state of the art are within implementation of RLNC for mobile devices. During this work several practical issues here identified and possible solutions put forward. In particular the focus was on reducing the complexity of RLNC in order to increase the coding throughput. Some of the insights has contributed to the development of the Kodo software library.

Commercially oriented activities have been the founding of the company Steinwurf, and projects conducted in cooperation with Nokia, Renesas, MIT, and NBC.

3.1 Practical Problems in Mobile Systems

When implementing ideas that appear straight forward in theory it is not uncommon that a series of unanswered questions and problems arise. Some can be trivial while others can expose fundamental problems, regardless they must be addressed before the concept can be utilized in practical systems.

Background

When this worked started some experience with practical systems based on RLNC was reported in the literature. However, the focus was primarily on designing distribution systems and protocols and therefore it appeared that several problems were ignored. As a consequence some of the ideas from theory were adopted unmodified [74] with unsatisfactory results [6, 73]. When and how recoding should be performed in practice were not clearly understood. An interesting recent example is [79] where recoding is not used, and thus the single most prominent feature of NC is forfeited, this example is not unique. Discarding the use of recoding degenerates the code to an end-to-end Forward Error Correction (FEC) code and removes most of the properties that makes NC codes useful in cooperative networks.

Contribution

In [25] we presented the first implementation of RLNC that utilized the binary field and took advantage of systematic coding, which demonstrated for the first time that RLNC is feasible on mobile devices. The binary field is directly supported by modern CPUs which simplifies implementation and makes RLNC practical on storage constraint devices such as sensor boards. We also presented an on-the-fly decoding algorithm which reduces the final decoding delay and distributes the decoding workload over time.

In [17] we explained why a pseudo random function cannot be used when recoding is supported and that it is therefore necessary to always include the coding vector. We showed that when the overhead due to the coding vector is included in the total overhead, it has a profound impact on the choice of suitable coding parameters. This enabled us to provide practical guidelines for the choice of coding parameters, based on the type of content.

In [1] we proposed a coding approach that mitigates the problem of representing the coding vector while preserving the possibility to recode. Furthermore, it is trivial to represent the resulting coding vectors in a near optimal way, from a compression point of view.

3.2 Reduced Coding Complexity

The coding complexity of RLNC defines the platforms on which the technique can be utilized. In particular, this is relevant for battery-driven devices as increased computational load leads to increased energy consumption.

Background

In the initial phase of this project, multiple implementations of RLNC had were presented [77, 78, 80–83] which focused on optimizing the underlying FFs, by parallelization or acceleration via GPUs. One likely reason was the predominate assumption that the size of the used field should be large to reduce the overhead due to linear dependency. Particular noteworthy efforts are those of [84] who focused on parallelization the coding both on CPUs and GPUs, and [85] who continued this work in addition to deployment on mobile phones. The problem of cross compatible use of GPUs remain, in particular on mobile platforms, but with the adoption of OpenCL [86] this could change in the future. Backed by Intel, Advanced Micro Devices, Nvidia, and ARM Holdings, OpenCL has been adopted by all relevant manufacturers in the PC and mobile spaces, and thus has the potential to become an unified solution on platforms from High Performance Computing (HPC) to mobile [87]. Recently new low complexity coding approaches have been proposed, e.g. [88] where degree distributions similar to those of Fountain codes are applied, or the modification of convolutional codes to [89]. These efforts are so far mostly theoretical whereas our work is conducted with a close coupling to implementation challenges. Additionally, our approach is different as it is based on non-uniform coding without a degree distribution.

Contribution

In [25] we introduced binary and systematic versions of RLNC. Binary codes are generally applicable at a slight cost in linear dependency, while systematic codes are only useful in typologies with up to two hops. The measured coding throughput was significantly higher than any previous implementation.

In [19] we presented several decoding algorithms intended for faster decoding of sparsely coded symbols. The problem of fill-in was observed.

In [1] we proposed an approach to coding that significantly increased the coding throughput, particularly at high generation sizes, which is where the performance of RLNC becomes problematic. This is achieved by coding non-uniformly over the original symbols which makes it possible to decode in a way that reduces the effect of fill-in significantly.

3.3 Dissemination

The Kodo Library

Work on the Kodo library, which implements NC algorithms and the underlying FFs started during the M.Sc. studies of Morten V. Pedersen and myself [90, 91]. During our Ph.D studies we have continued to contribute to Kodo by using it to test multiple ideas for improving the speed of RLNC and supporting various new features.

To enable others to experiment with RLNC we have made Kodo available under a license that permits the use of Kodo for research and teaching purposes. By doing so we hope that others will use Kodo and potentially contribute optimizations or ideas back in the future. Additionally it allow other researchers to verify the measurements we conduct with Kodo, and ensure maximum transparency. Kodo has been used in multiple student projects at Aalborg University. Additionally it is currently used by research groups in Hong Kong, Portugal and Hungary, but as it is freely available it could be in more widespread use [92].

Steinwurf ApS

In 2011 Morten V. Pedersen, Frank H.P. Fitzek, Muriel Medard and myself founded Steinwurf. Steinwurf conducts the continued development of Kodo and the effort to commercialize the software. In addition, Steinwurf is developing a series of software libraries, e.g. for cross-platform deployment, network integration and reliability protocols. These components are necessary in order to utilize RLNC efficiently in a real world application, and to face the many practical challenges that arises outside of purpose built networks constructed in the laboratory.

Steinwurf also develops mobile phone applications, primarily inspired by our early work with local content distribution and local cooperation [26]. Local sharing of pictures to multiple sinks, and streaming of audio and video to multiple sinks. These applications are intended to showcase the potential of NC technology and the capability of the Kodo library and related software.

ENOC and NOCE projects

The Evolved Network COding (ENOC) project in cooperation with Nokia started shortly after the COoperation and NEtwork coding (CONE) project where this work was carried out. It later continued as the Network COding Evolved (NOCE) project in cooperation with Renesas. The aim of the project was to investigate the use of NC in wireless systems and create demonstrators of the proposed systems. Some of the outcome was published in the following papers.

[2] Peter Vingelmann, Janus Heide, Morten V. Pedersen, Qi Zhang, and Frank H. P. Fitzek. “Decentralized Data Dissemination with Network Coding in High-Mobility MANETs”. In: *IEEE Transaction on Mobile Computing* (2012). submitted.

[4] Peter Vingelmann, Frank H.P. Fitzek, Morten V. Pedersen, Janus Heide, and Hassan Charaf. “Synchronized Multimedia Streaming on the iPhone Platform with Network Coding”. In: *IEEE Communications Magazine - Consumer Communications and Networking Series* 49.6 (2011), pp. 126–132.

[12] Péter Vingelmann, Morten V. Pedersen Janus Heide, Qi Zhang, and Frank H.P. Fitzek. “Data Dissemination in the Wild: A Testbed for High-Mobility MANETs”. In: *IEEE International Conference on Communications (ICC) - Wireless Networking Symposium.* Ottawa, Canda, June 10–15, 2012.

[14] Péter Vingelmann, Morten V. Pedersen, Frank H.P. Fitzek, and Janus Heide. “On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices”. In: *IEEE GLOBal COMmunication conference, exhibition & industry forrum (GLOBECOM) - Next Generation Networking Symposium.* Houston, Texas, USA, Dec. 5–9, 2011.

[21] Peter Vingelmann, Frank H.P. Fitzek, Morten V. Pedersen, Janus Heide, and Hassan Charaf. “Synchronized Multimedia Streaming on the iPhone Platform with Network Coding.” In: *IEEE Consumer Communications and Networking Conference (CCNC) - Multimedia & Entertainment Networking and Services Track.* **Best student paper award**. Las Vegas, NV, USA, Jan. 9–12, 2011.

[22] Morten V. Pedersen, Janus Heide, Peter Vingelmann, Laszlo Blazovics, and Frank H.P. Fitzek. “Multimedia Cross–Platform Content Distribution for Mobile Peer–to–Peer Networks using Network Coding”. In: *ACM Multimedia - Multimedia Applications Track Short Paper.* Firenze, Italy, Oct. 25–29, 2010.

[23] Peter Vingelmann, Morten V. Pedersen, Frank H.P. Fitzek, and Janus Heide. “Multimedia Distribution using Network Coding on the iPhone Platform”. In: *ACM Multimedia - ACM Workshop on Mobile Cloud Media Computing.* Firenze, Italy, Oct. 29, 2010.

NBC project

In cooperation with the group of Muriel Medard we developed a demonstrator, that showed the possibility of distributing live video streams in a P2P fashion. The developed testbed was demonstrated at a meeting at NBC's headquarters in New York, US. This provided unique feedback on an important business players perspective on the technology and the future of distribution networks.

Chapter 4

Conclusion

The work in this thesis has addressed problems related to deploying RLNC on mobile devices such as mobile phones and tablets. The aim was to enable practical systems for mobile devices based on RLNC to be built. The work has been targeted toward mobile devices as the popularity of such devices is quickly rising and they may become the dominant platform in the near future. This class of devices also pose additional challenges in terms of computational capability and energy compared to other more powerful computer devices.

To increase the coding throughput and decrease the energy consumption we proposed to simplify RLNC by decreasing the field size and coding systematically. A small field can be used in many networks and unless the generation is very small this approach introduces a modest overhead as a consequence of higher probability of linearly dependent symbols. Systematic codes are not as generally applicable as they are only useful on the first hop from the source and therefore less interesting in multihop scenarios.

To be able to signal information about the state of the decoding we proposed to embed information into the coding vector. This information can be used to signal information about the state of the decoding and thus enable encoding nodes to code in a way such that decoding becomes simpler. We enabled this by observing that operations performed in the binary field are also valid operations in any binary extension field and thus information can be embedded by drawing random elements from a specific field. However, we have not pursued this further as it requires that a big field is used.

We considered using a sparse variant of RLNC in order to increase the coding throughput. To understand the implications of doing so we analyzed the resulting overhead. We found that if the density is chosen correctly an overhead arbitrarily close to that of dense RLNC can be obtained. As the generation size grows the good choice of density decreases, which makes such an approach most useful for applications with relaxed delay requirements.

When decoding sparse symbols we found that the throughput did not increase significantly when existing RLNC algorithms were used. The reason was the effect of fill-in and we considered several new decoding approaches and algorithms, designed to decrease the number of operations performed when decoding sparsely coded symbols. When testing these algorithms we found that they generally provided a slightly lower decoding complexity, also when decoding dense symbols. However, the reduction in complexity was not sufficient to increase the coding throughput significantly.

We also showed that a conflict exists between using the commonly assumed practice of compressing coding vectors using a pseudo random function and supporting recoding. This became apparent as we investigated sparse codes where the combinations of a set of sparse symbols with high probability is less sparse than the individual combined symbols. This insight has the implication that another representation of the coding vector must be used. In the case of dense codes using a high field size this is non-trivial or impossible as the entropy of the coding vector is high. This underlines that in many cases the choice of suitable coding parameters does not degenerates to the biggest field and generation size that can be supported.

Finally, we proposed an approach that addresses the introduced issues. Instead of combining all symbols at random only neighboring symbols are combined. This makes it possible to perform decoding efficiently without the effect of fill-in and without using overly complicated decoding algorithms. At the same time it makes it trivial to create compact representations of the coding vector. It also enables an overhead due to linearly dependent symbols arbitrarily close to that of RLNC. Importantly, this approach makes it possible to perform recoding and we have introduced several new approaches to do so. To verify the gain of this approach in terms of coding throughput it was implemented in C++ and compared to a similar standard RLNC implementation. For the largest generation tested, which was 2048, the gain in terms of both encoding and decoding was approximately an order of magnitude.

The knowledge obtained through this project has contributed to the development and improvement of the Kodo software library. I have also been involved in several projects where the demonstrators and publications were produced. Additionally, the company Steinwurf was founded to continue the development and commercializing of Kodo.

As a closing remark we note that computer networks are bound to go through a paradigm shift some time in the future, it has happened before and it will happen again. If this occur in the near future NC could become an import part of the solution. If so this thesis will hopefully add a small contribution to the field of RLNC implementation and help propel RLNC towards real world adoption and deployment.

Complete List of Publications

Journals & Magazines

[1] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Muriel Médard. "Perpetual Codes for Network Coding". In: *IEEE Transaction on Mobile Computing* (2012). submitted.

[2] Peter Vingelmann, Janus Heide, Morten V. Pedersen, Qi Zhang, and Frank H. P. Fitzek. "Decentralized Data Dissemination with Network Coding in High-Mobility MANETs". In: *IEEE Transaction on Mobile Computing* (2012). submitted.

[3] Frank H.P. Fitzek, Janus Heide, Morten V. Pedersen, and Marcos Katz. "Implementation of Network Coding for Social Mobile Clouds". In: *IEEE Signal Processing Magazine* (2012). accepted.

[4] Peter Vingelmann, Frank H.P. Fitzek, Morten V. Pedersen, Janus Heide, and Hassan Charaf. "Synchronized Multimedia Streaming on the iPhone Platform with Network Coding". In: *IEEE Communications Magazine - Consumer Communications and Networking Series* 49.6 (2011), pp. 126–132.

[5] Morten V. Pedersen, Janus Heide, Frank H.P. Fitzek, and Torben Larsen. "A Mobile Application Prototype using Network Coding". In: *European Transactions on Telecommunications (ETT)* 21.8 (8 2010), 738–749.

[6] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Torben Larsen. "Cautious View on Network Coding - From Theory to Practice". In: *Journal of Communications and Networks (JCN)* 10.4 (2008), pp. 403–411.

Conference Proceedings

[7] Achuthan Paramanathan, Janus Heide, Peyman Pahlavani, Martin Hundebøll, Stephan Alexander Rein, Frank H.P. Fitzek, and Gergő Ertli. “Energy and Data Throughput for Asymmetric Inter-Session Network Coding”. In: *IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD).* accepted. Barcelona, Spain, Sept. 17–19, 2012.

[8] Janus Heide, Péter Vingelmann, Morten V. Pedersen, Qi Zhang, and Frank H.P. Fitzek. “The Impact of Packet Loss Behavior in 802.11 b/g on the Cooperation Gain in Reliable Multicast”. In: *IEEE Vehicular Technology Conference (VTC) - Wireless networks Symposium.* accepted. Québec, Canada, Sept. 3–6, 2012.

[9] Martin Hundebøll, Jeppe Leddet-Pedersen, Janus Heide, Morten V. Pedersen, Stephan A. Rein, and Frank H.P. Fitzek. “CATWOMAN: Implementation and Performance Evaluation of IEEE 802.11 based Multi-Hop Networks using Network Coding”. In: *IEEE Vehicular Technology Conference (VTC) - Ad-Hoc, mesh and sensor networks.* accepted. Québec, Canada, Sept. 3–6, 2012.

[10] Peyman Pahlevani, Achuthan Paramanathan, Martin Hundebøll, Janus Heide, Stephan A. Rein, and Frank H.P. Fitzek. “Reliable Communication in Wireless Meshed Networks using Network Coding”. In: *IEEE Vehicular Technology Conference (VTC) - Wireless networks Symposium.* accepted. Québec, Canada, Sept. 3–6, 2012.

[11] Raúl Palacios, Janus Heide, Frank H.P. Fitzek, and Fabrizio Granelli. “Design and Performance Evaluation of Underwater Data Dissemination Strategies using Interference Avoidance and Network Coding”. In: *IEEE International Conference on Communications (ICC) - Communication QoS, Reliability and Modeling Symposium.* Ottawa, Canda, June 10–15, 2012.

[12] Péter Vingelmann, Morten V. Pedersen Janus Heide, Qi Zhang, and Frank H.P. Fitzek. “Data Dissemination in the Wild: A Testbed for High-Mobility MANETs”. In: *IEEE International Conference on Communications (ICC) - Wireless Networking Symposium.* Ottawa, Canda, June 10–15, 2012.

[13] Qi Zhang, Janus Heide, Morten V. Pedersen, and Frank H.P. Fitzek. “User Cooperation with Network Coding for MBMS”. In: *IEEE GLOBal COMmunication conference, exhibition & industry forrum (GLOBE-*

COM) - Communication Software, Services, and Multimedia Applications Symposium. Houston, Texas, USA, Dec. 5–9, 2011.

[14] Péter Vingelmann, Morten V. Pedersen, Frank H.P. Fitzek, and Janus Heide. "On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices". In: *IEEE GLOBal COMmunication conference, exhibition & industry forrum (GLOBECOM) - Next Generation Networking Symposium.* Houston, Texas, USA, Dec. 5–9, 2011.

[15] Janus Heide, Qi Zhang, Morten V. Pedersen, and Frank H.P. Fitzek. "Reducing Computational Overhead of Network Coding with Intrinsic Information Conveying". In: *IEEE Vehicular Technology Conference (VTC) - Transmission Technologies Track.* San Francisco, CA, USA, Sept. 5–8, 2011.

[16] Frank H.P. Fitzek, Janus Heide, and Morten V. Pedersen. "On the Need of Network coding for Mobile Clouds". In: *Automation and Applied Computer Science Workshop (AACS).* Budapest, Hungary, June 24, 2011.

[17] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Muriel Médard. "On Code Parameters and Coding Vector Representation for Practical RLNC". In: *IEEE International Conference on Communications (ICC) - Communication Theory Symposium.* Kyoto, Japan, June 5–9, 2011.

[18] Frank H.P. Fitzek, Janus Heide, Morten V. Pedersen, Gergö Ertli, and Marcos Katz. "Multi-Hop versus Overlay Networks: A Realistic Comparison Based on Energy Requirements and Latency". In: *IEEE Vehicular Technology Conference (VTC) - Workshop on Cognitive radio and Cooperative strategies for POWER saving (C2Power).* Budapest, Hungary, May 15–18, 2011.

[19] Janus Heide, Morten V. Pedersen, and Frank H.P. Fitzek. "Decoding Algorithms for Random Linear Network Codes". In: *IFIP International Conferences on Networking - Workshop on Network Coding Applications and Protocols (NC-Pro).* Vol. 6827. Lecture Notes in Computer Science. Valencia, Spain, May 12, 2011, pp. 129–137.

[20] Morten V. Pedersen, Janus Heide, and Frank H.P. Fitzek. "Kodo: An Open and Research Oriented Network Coding Library". In: *IFIP International Conferences on Networking - Workshop on Network Coding Applications and Protocols (NC-Pro).* Vol. 6827. Lecture Notes in Computer Science. Valencia, Spain, May 12, 2011, pp. 145–153.

[21] Peter Vingelmann, Frank H.P. Fitzek, Morten V. Pedersen, Janus Heide, and Hassan Charaf. “Synchronized Multimedia Streaming on the iPhone Platform with Network Coding.” In: *IEEE Consumer Communications and Networking Conference (CCNC) - Multimedia & Entertainment Networking and Services Track*. **Best student paper award**. Las Vegas, NV, USA, Jan. 9–12, 2011.

[22] Morten V. Pedersen, Janus Heide, Peter Vingelmann, Laszlo Blazovics, and Frank H.P. Fitzek. “Multimedia Cross–Platform Content Distribution for Mobile Peer–to–Peer Networks using Network Coding”. In: *ACM Multimedia - Multimedia Applications Track Short Paper*. Firenze, Italy, Oct. 25–29, 2010.

[23] Peter Vingelmann, Morten V. Pedersen, Frank H.P. Fitzek, and Janus Heide. “Multimedia Distribution using Network Coding on the iPhone Platform”. In: *ACM Multimedia - ACM Workshop on Mobile Cloud Media Computing*. Firenze, Italy, Oct. 29, 2010.

[24] Frank H.P. Fitzek, Morten V. Pedersen, Janus Heide, and Muriel Médard. “Network Coding: Applications and Implementations on Mobile Devices”. In: *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. Bodrum, Turkey, Oct. 17–21, 2010.

[25] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Torben Larsen. “Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes”. In: *IEEE International Conference on Communications (ICC) - Workshop on Cooperative Mobile Networks*. Dresden, Germany, June 14–18, 2009.

[26] Morten V. Pedersen, Janus Heide, Frank H.P. Fitzek, and Torben Larsen. “PictureViewer - A Mobile Application using Network Coding”. In: *European Wireless Conference (EW)*. Aalborg, Denmark, May 17–20, 2009.

[27] Kasper Revsbech, Janus Heide, Kim Højgaard-Hansen, Gian Paolo Perrucci, and Frank H.P. Fitzek. “Energy Saving Potential Using Active Networking on Linux Phones”. In: *European Wireless Conference (EW)*. Aalborg, Denmark, May 17–20, 2009.

[28] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Torben Larsen. “Connecting the Islands - Enabling Global Connectivity through Local Cooperation”. In: *International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (Mobilware) - International Workshop on Interconnecting Ubiquitous Islands*

using Mobile and Next Generation Networks (Ubi-Islands). Berlin, Germany, Apr. 28–30, 2009.

[29] Janus Heide, Jesper H. Sørensen, Rasmus Krigslund, Petar Popovski, Jacob Chakareski, and Torben Larsen. “Cooperative Video Streaming with Dynamic Compression at the Terminal Nodes”. In: *IEEE International Symposium on Image/Video Communications over fixed and mobile networks (ISIVC)*. Bilbao, Spain, July 9–11, 2008.

[30] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, Tatiana V. Kozlova, and Torben Larsen. “Know Your Neighbour: Packet Loss Correlation in IEEE 802.11b/g Multicast”. In: *The International Mobile Multimedia Communications Conference (MobiMedia)*. Oulu, Finland, July 7–9, 2008.

[31] Janus Heide, Jesper H. Sørensen, Rasmus Krigslund, Petar Popovski, Jacob Chakareski, and Torben Larsen. “Cooperative Media Streaming using Adaptive Network Compression”. In: *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) - IEEE Workshop on Mobile Video Delivery (MoViD)*. Newport Beach, CA, USA, June 23–26, 2008, pp. 1–7.

[32] Brian A. Mertz, Janus Heide, Jesper H. Sørensen, Rasmus Krigslund, and Simon J. K. Pedersen. “Communication Beyond (N)ACKs: Wireless Transmission with Informative Feedback”. In: *The Annual IEEE Student Paper Conference (AISPC)*. Aalborg, Denmark, Feb. 15, 2008, pp. 1–4.

Poster Presentations

[33] Morten V. Pedersen, Janus Heide, Peter Vingelmann, Leonardo Militano, and Frank H.P. Fitzek. “Network Coding on Mobile Devices”. In: *Workshop on Network Coding, Theory and Applications (NetCod)*. Lausanne, Switzerland, June 15–16, 2009.

Book Chapters

[34] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Torben Larsen. “Network Coding in the Real World”. In: *Network Coding: Fundamentals and Applications*. Ed. by Muriel Medard and Alex Sprintson. Academic Press, Oct. 17, 2011. Chap. 4, pp. 87–114.

[35] Qi Zhang, Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, Jorma Lilleberg, and Kari Rikkinen. "Network Coding and User Cooperation for Streaming and Download Services in LTE Networks". In: *Network Coding: Fundamentals and Applications*. Ed. by Muriel Medard and Alex Sprintson. Academic Press, Oct. 17, 2011. Chap. 5, pp. 115–140.

[36] Morten V. Pedersen, Janus Heide, Frank H.P. Fitzek, and Tony Torp. "Getting Started". In: *Qt for Symbian*. Ed. by Frank H.P. Fitzek and Tony Torp. John Wiley & Sons, Ltd, 2010, pp. 13–28.

[37] Janus Heide and Leonardo Militano. "Introduction to Network Coding for Mobile Peer to Peer". In: *Mobile Peer to Peer (P2P)*. Ed. by Frank H.P Fitzek and Hassan Charaf. John Wiley & Sons, Ltd, 2009, pp. 143–159.

Patent Applications

[38] Frank H.P. Fitzek, Janus Heide, and Morten V. Pedersen. "Network Coding by Beam Forming". Pat. PCT/IB2011/000362, pending. Feb. 22, 2011.

[39] Janus Heide, Morten V. Pedersen, Frank H.P. Fitzek, and Qi Zhang. "Intrinsic Information Conveying in Network Coding". Pat. 800.0406.U1 (US), pending. Mar. 19, 2010.

References

[40] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. “Network information flow”. In: *IEEE Transactions on Information Theory* 46.4 (2000), pp. 1204–1216.

[41] R. W. Yeung and Zhen Zhang. “Distributed source coding for satellite communications”. In: *Information Theory, IEEE Transactions on* 45.4 (1999), pp. 1111–1120. URL: http://dx.doi.org/10.1109/18.761254.

[42] P. Elias, A. Feinstein, and C. E. Shannon. “Note on Maximum Flow Through a Network”. In: *IRE Transactions on Information Theory IT-2* (1956), pp. 117–199.

[43] Petar Popovski and Hiroyuki Yomo. “Bi-directional Amplification of Throughput in a Wireless Multi-Hop Network.” In: *VTC Spring*. Melbourne, Australia: IEEE, 2006, pp. 588–593. URL: http://dblp.uni-trier.de/db/conf/vtc/vtc2006s.html#PopovskiY06.

[44] 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)*. Pisa, Italy: ACM Press, 2006, pp. 243–254.

[45] Sachin Katti, Dina Katabi, Wenjun Hu, Hariharan Rahul, and Muriel Medard. “The importance of being opportunistic: Practical network coding for wireless environments”. In: *In Proceedings of 43rd Allerton Conference on Communication, Control, and Computing*. 2005.

[46] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Michael Mitzenmacher, and João Barros. “Network coding meets TCP”. In: *CoRR* abs/0809.5022 (2008).

[47] C. Gkantsidis and P. R. Rodriguez. “Network coding for large scale content distribution”. In: *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 4. 2005, pp. 2235–2245. URL: http://dx.doi.org/10.1109/INFCOM.2005.1498511.

[48] Larissa Popova, Armin Schmidt, Wolfgang Gerstacker, and Wolfgang Koch. “Network Coding Assisted Mobile-To-Mobile File Transfer”. In: *Australasian Telecommunication Networks and Applications Conference (ATNAC)*. Christchurch, New Zealand, 2007.

[49] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. “Trading structure for randomness in wireless opportunistic routing”. In: *SIGCOMM ’07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. Kyoto, Japan: ACM, 2007, pp. 169–180.

[50] Sachin Katti, Dina Katabi, Hari Balakrishnan, and Muriel Medard. “Symbol-Level Network Coding for Wireless Mesh Networks”. In: *ACM SIGCOMM*. Seattle, WA, 2008.

[51] Mea Wang and Baochun Li. “Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming”. In: *INFOCOM*. May 2007, pp. 1082–1090.

[52] Morten V. Pedersen, Frank H.P. Fitzek, and Torben Larsen. “Implementation and Performance Evaluation of Network Coding for Cooperative Mobile Devices”. In: *IEEE Cognitive and Cooperative Wireless Networks Workshop*. Beijing, China: IEEE, 2008.

[53] D. J. C. MacKay. “Fountain codes”. In: *Communications, IEE Proceedings-* 152.6 (Dec. 2005), pp. 1062–1068. URL: http://dx.doi.org/10.1049/ip-com:20050237.

[54] Michael Luby. “LT Codes”. In: *Proceedings of the 43rd Symposium on Foundations of Computer Science*. FOCS ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 271–. URL: http://dl.acm.org/citation.cfm?id=645413.652135.

[55] A. Shokrollahi. “Raptor codes”. In: *IEEE Transactions on Information Theory* 52.6 (June 2006), pp. 2551–2567.

[56] Cisco. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011–2016*. Tech. rep. 2012. URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

[57] F.H.P. Fitzek and M. Katz, eds. *Cooperation in Wireless Networks: Principles and Applications – Real Egoistic Behavior is to Cooperate!* Springer, 2006.

[58] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes". In: *IEEE Communications Surveys and Tutorials* 7 (2005), pp. 72–93.

[59] F.H.P. Fitzek and H. Charaf. *Mobile Peer to Peer (P2P): A Tutorial Guide*. Wiley Series in Communications Networking & Distributed Systems. John Wiley & Sons, 2009. URL: http://books.google.dk/books?id=973_bs-by3kC.

[60] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. "Wireless mesh networks: a survey". In: *Comput. Netw. ISDN Syst.* 47.4 (Mar. 2005), pp. 445–487. URL: http://dx.doi.org/10.1016/j.comnet.2004.12.001.

[61] Changling Liu and Jörg Kaiser. *A Survey of Mobile Ad Hoc network Routing Protocols*. Tech. rep. 2003-08. University of Ulm, Oct. 2003. URL: http://vts.uni-ulm.de/docs/2005/5346/vts_5346.pdf.

[62] Nokia. *N70 hardware specifications*. 2005. URL: http://www.gsmarena.com/nokia_n70-1153.php.

[63] Nokia. *N95 hardware specifications*. 2007. URL: http://www.gsmarena.com/nokia_n95-1716.php.

[64] Apple. *iPhone hardware specifications*. 2007. URL: http://www.gsmarena.com/apple_iphone-1827.php.

[65] Toshiba. *TG01 hardware specifications*. 2009. URL: http://www.gsmarena.com/toshiba_tg01-2662.php.

[66] LG. *Optimus 2X hardware specifications*. 2011. URL: http://www.gsmarena.com/lg_optimus_2x-3598.php.

[67] HTC. *One X hardware specifications*. 2012. URL: http://www.gsmarena.com/htc_one_x-4320.php.

[68] 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*. Pacifico Yokohama, Kanagawa, Japan, 2003. URL: citeseer.ist.psu.edu/ho03benefits.html.

[69] Shuo yen Robert Li, Raymond W. Yeung, and Ning Cai. “Linear network coding”. In: *IEEE Transactions on Information Theory* 49 (2003), pp. 371–381.

[70] Philip A. Chou, Yunnan Wu, and Kamal Jain. “Practical Network Coding”. In: *Proceedings of the annual Allerton conference on communication control and computing* 4 (2003), pp. 40–49.

[71] P. Maymounkov, N. J. A. Harvey, and D. S. Lun. “Methods for Efficient Network Coding”. In: *44th Allerton Annual Conference* (2006).

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

[73] Mea Wang and Baochun Li. “How Practical is Network Coding?” In: *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on* (2006), pp. 274–278.

[74] C. Fragouli, J.L. Boudec, and J. Widmer. “Network coding: an instant primer”. In: *SIGCOMM Comput. Commun. Rev.* 36.1 (2006), pp. 63–68.

[75] Christos Gkantsidis, John Miller, and Pablo Rodriguez. “Comprehensive view of a live network coding P2P system”. In: *IMC ’06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. Rio de Janeriro, Brazil: ACM, 2006, pp. 177–188. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=69452.

[76] Christos Gkantsidis, John Miller, and Pablo Rodriguez. “Anatomy of a P2P Content Distribution system with Network Coding”. In: Santa Barbara, CA, USA, 2006. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=67453.

[77] Hassan Shojania and Baochun Li. “Parallelized Progressive Network Coding With Hardware Acceleration”. In: *Quality of Service, 2007 Fifteenth IEEE International Workshop on*. 2007, pp. 47–55.

[78] Xiaowen Chu, Kaiyong Zhao, and Mea Wang. “Massively Parallel Network Coding on GPUs”. In: *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*. 2008, pp. 144–151.

[79] Zimu Liu, Chuan Wu, Baochun Li, and Shuqiao Zhao. “UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding”. In: *IEEE International Conference on Computer Communications*. San Diego, CA, 2010, pp. 2070–2078.

[80] Hassan Shojania, Baochun Li, and Xin Wang. "Nuclei: GPU-accelerated Many-core Network Coding". In: *The 28th Conference on Computer Communications (INFOCOM 2009)*. Rio de Janeiro, Brazil, 2009.

[81] Peter Vingelmann, Peter Zanaty, Frank H.P. Fitzek, and Hassan Charaf. "Implementation of Random Linear Network Coding on OpenGL-enabled Graphics Cards". In: *European Wireless 2009*. Aalborg, Denmark, 2009.

[82] Hassan Shojania and Baochun Li. "Pushing the Envelope: Extreme Network Coding on the GPU". In: *Distributed Computing Systems, International Conference on* 0 (2009), pp. 490–499.

[83] Péter Vingelmann and Frank H. P. Fitzek. "Implementation of Random Linear Network Coding Using NVIDIA's CUDA Toolkit". In: *Networks for Grid Applications - Third International ICST Conference*. Athens, Greece, 2009.

[84] Mea Wang. "A Quest for High-Performance Peer-to-Peer Live Streaming". PhD thesis. Graduate Department of Electrical and Computer Engineering University of Toronto, 2008. URL: http://www.cpsc.ucalgary.ca/~meawang/papers/mwang-PhD.pdf.

[85] Hassan Shojania. "Making Coding Practical: From Servers to Smartphones". PhD thesis. Graduate Department of Electrical and Computer Engineering University of Toronto, 2010. URL: http://hassan.shojania.com/thesis/PhD_thesis.pdf.

[86] Khronos Group. *OpenCL specifications and materials*. 2012. URL: http://www.khronos.org/opencl/.

[87] HPC wire. *OpenCL Gains Ground On CUDA*. 2012. URL: http://www.hpcwire.com/hpcwire/2012-02-28/opencl_gains_ground_on_cuda.html.

[88] Shenghao Yang and R.W. Yeung. "Coding for a network coded fountain". In: *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*. 2011, pp. 2647 –2651.

[89] Vladimir Sidorenko Antonia Wachter-Zeh. "Rank Metric Convolutional Codes for Random Linear Network Coding". In: *IEEE International Symposium on Network Coding 2012 (Netcod)*. Boston, USA, 2012.

[90] Morten V. Pedersen and Janus Heide. "Creating Network Coding Potential for Cooperating Mobile Devices". Technical report. Aalborg University, 2008.

[91] Morten V. Pedersen and Janus Heide. “Performance Evaluation of Cooperation in Wireless Networks”. M.Sc. thesis. Aalborg University, 2009.

[92] Steinwurf. *The Kodo library.* 2012. URL: https://github.com/steinwurf/kodo.

Chapter 5

Enclosed Papers

Paper 1

Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes

Janus Heide
Morten V. Pedersen
Frank H.P. Fitzek
Torben Larsen

IEEE International Conference on Communications (ICC)
Workshop on Cooperative Mobile Networks
Dresden, Germany, June 14–18, 2009

Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes

Janus Heide
Dept. of Electronic Systems
Aalborg University
Email: speje@es.aau.dk

Morten V. Pedersen
Dept. of Electronic Systems
Aalborg University
Email: mvpe@es.aau.dk

Frank H. P. Fitzek
Dept. of Electronic Systems
Aalborg University
Email: ff@es.aau.dk

Torben Larsen
Dept. of Electronic Systems
Aalborg University
Email: tl@es.aau.dk

***Abstract*—In this work we consider the implementation of Random Linear Network Coding (RLNC) on battery constrained mobile devices with low computational capabilities such as; sensors, mobile phones and Personal Digital Assistants (PDAs). It is non-trivial to create an efficient implementation of RLNC which is needed to ensure high throughput, low computational requirements and energy consumption. As a consequence there does not, to the best of our knowledge, exist any such implementation for mobile device that allow for throughput close to what can be achieved in e.g. Wireless Local Area Network (WLAN).**

In this paper we propose to base RLNC on the binary Galois field and to use a systematic code. We have implemented this approach in C++ and Symbian C++ and achieve synthetic encoding/decoding throughput of up to 40/30 MB/s on a Nokia N95-8GB mobile phone and 1.5/1.0 GB/s on a high end laptop.

***Index Terms*—Mobile devices, Network coding, Reliable Multicast.**

I. Introduction

A large body of existing literature [1] treats the theoretical benefits of Network Coding (NC). However, the costs of implementing NC in terms computational overhead, memory consumption or network usage is often not considered. In this work we consider the implementation of RLNC on mobile battery constrained devices with low computational capabilities, such as sensors, mobile phones or PDAs. The computations performed using RLNC is based on finite fields arithmetic also known as Galois fields. From a coding perspective the field size, q, used should be large to ensure that coded packets are linearly independent, additionally increasing the size of the field elements is advantageous as this reduces the number of operations needed to code a certain amount of data [2]. However as the field size grows it becomes difficult to construct an efficient implementation of the necessary operations. Although the attention towards practical implementation of NC has been increased [3], [4], [5], some issues remains to be solved in order to pave the way for successful deployment of NC.

In [6], [7] implementation problems are investigated with focus on the computational complexity of coding operations. An implementation using log and anti-log tables is constructed and evaluated. Throughput up to 11 MB/s is measured on a 3.6 GHz dual core Intel P4 Central Processing Unit (CPU), but performance decreases rapidly as the number of packets that are coded together increases. Different optimization techniques are described and incorporated. Generated network overhead as a consequence of utilizing NC is commented and sought to be minimized. Furthermore the concepts of *aggressiveness* and *density* and its impact of performance are presented. In [8] a coding throughput of 44Mb/s is measured using an implementation based on a simple full-size look-up table. This is achieved on an 800 MHz Intel Celeron CPU when 32 packets, *g=32*, of 1500 B are coded together. The observed throughput is approximately 10 times higher than that reported in [6] at similar settings. The authors conclude that deployment is not a problem, however it is mentioned that throughput for *g=32* is the best case, unfortunately no performance for *g>32* is presented. In [9] tables are not used for Galois field multiplication, instead the authors implement a loop based approach. In combination with Single Instruction, Multiple Data (SIMD) instructions this approach provides a performance increase of over 500% compared to a baseline implementation. The implementation is further optimized through parallelization and encoding speeds up to 43 MB/s is measured on a 2.8 GHz quad core Intel P4 CPU. Although these results show promising coding throughputs most were achieved at maximum CPU utilization, which is not acceptable on non-dedicated machines. Additionally these results are not in general valid for mobile phones or wireless sensors, as the hardware resources available on such devices are considerably lower than those of the desktop computers used to obtain these results. In [2] we present the results of a basic log and anti-log table implementation running on a Nokia N95 with a 332 MHz ARM 11 CPU. We achieved the maximum coding throughput of 117 KB/s for a GF(2^{16}), which indicates that before NC can be deployed on mobile devices, more efficient algorithms or hardware acceleration is needed.

In this work we propose to use the binary field to reduce the computational complexity of RLNC and to use a systematic [10] approach to reduce the amount of coding needed. Furthermore we provide an analysis of this approach for different generation sizes and compare its performance to network coding with a higher field size.

This work is organized in the following sections. Section III describes the scenario under investigation. Section II presents an analysis of the proposed network coding approach. In Section IV we introduce our implementation using GF(2) and coding throughputs obtained with this implementation. The final conclusions are drawn in Section V.

II. Scenario & Solution

We consider a scenario where a source s wants to reliably transmit the same data to one or more nearby sinks $t_1, t_2, \ldots, t_N$ via a wireless link. This basic scenario is given in Figure 1.

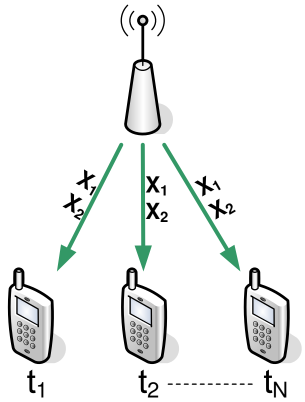


Fig. 1: One source s transmitting data to N receivers $t_1, t_2, \ldots, t_N$.

As all receivers are requesting the same data, broadcast provides an efficient utilization of the wireless channel, as all packets are delivered to all nodes simultaneously. To ensure that the links are reliable some form of retransmission is required, to correct packet losses at the individual nodes. In current networks this is done by letting the individual nodes acknowledged received packets. Thus retransmissions of multiple different packet sets are required unless all nodes lose exactly the same packets.

To ensure reliability with a low overhead we will instead use RLNC to correct packet errors. This is done by transmitting the data in two stages, in the first stage the source, s, transmits all packets uncoded. This makes sense as all packets received by the individual nodes will contain useful new information. In the second stage we wish to correct packet losses which have occurred during the first stage. Due to the uncorrelated nature of packet losses the nodes will now hold disjoint sets of packets. Therefore to maximize the number of nodes for which a packet is useful, the source will create and send random linearly combinations of the original data. By using this approach one coded packet carries information which can potentially correct different errors at different nodes simultaneously. To retrieve the full data set a node now has to receive as many linear independent coded packets as it lost during the first stage.

III. Analysis

In this section we analyze single-source multiple-sinks reliable transmission using Markov chains. The main objective is to determine the expected number of transmissions $E[t_x]$ needed for transmitting a packet from a source s to N sinks $t_1, t_2, \ldots t_N$. We assume an i.i.d. Packet Error Probability (PEP) p and consider unicast, broadcast, pure and systematic network coding. We note that the analysis also holds for unicast and RLNC when the channels are not independent, but that broadcast will perform better if the erasures are correlated. Unicast is not designed for this type of transmission however it is interesting as a reference.

The state machine of a single sink that receive one packet is illustrated on Figure 2. Either the node has received the packet and is in state zero, or it has not received the packet and is in state one. The number of the state thus indicates the number of erroneous or missing packets.

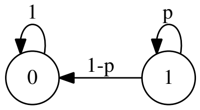


Fig. 2: Markov chain for a single node receiving a single packet

In the transmission matrix state zero corresponds to column zero and state one to column one. When s transmits the packet t_1 will receive it with probability $1-p$ and not receive it with probability p. This yields the transition matrix $\mathbf{T}$. When transmission commences the node have not received the packet and thus is in state one. It is convenient to define a matrix that indicates the starting probabilities, $\mathbf{S}$.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 \\ 1-p & p \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

A. Unicast

The probability distribution of a sink after i transmissions from s is given by the last row in $\mathbf{P}^i$, in this case row one denoted by $\mathbf{P}^i_{(1,:)}$, where.

$$\mathbf{P}^i = \mathbf{S} \times \mathbf{T}^i \tag{1}$$

The probability that the sink has received the packet after i transmissions is $\mathbf{P}^i_{(1,0)}$ which is the index row one, column zero in $\mathbf{P}^i$. Thus the probability that the sink has not received the packet is $1 - \mathbf{P}^i_{(1,0)}$. In unicast transmission is performed serially and thus we multiply with the number of sinks N.

$$E[t_x] = N \cdot \sum_{i=0}^{\infty} 1 - \mathbf{P}^i_{(1,0)} \tag{2}$$

B. Broadcast

In broadcast the probability distribution for one node is the same as for unicast but all nodes receive packets in parallel. Thus the probability that all sinks have received the packet after i transmissions is the probability that one node have received the packet to the power of the number of sinks, $(\mathbf{P}^i_{(1,0)})^N$.

$$E[t_x] = \sum_{i=0}^{\infty} 1 - \left(\mathbf{P}^i_{(1,0)}\right)^N \tag{3}$$

C. Pure Network Coding

In network coding 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 refereed 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 $\mathbf{M} = [\mathbf{m}_1\mathbf{m}_2 \dots \mathbf{m}_g]$, where $\mathbf{m}_i$ is a column vector.

In pure network coding to generate one coded data packet $\mathbf{x}$, $\mathbf{M}$ is multiplied with a randomly generated vector $\mathbf{g}$ of length g, $\mathbf{x} = \mathbf{M} \times \mathbf{g}$. In this way we can construct $\mathbf{X} = [\mathbf{x}_1\mathbf{x}_2 \dots \mathbf{x}_{g+r}]$ that consists of $g+r$ coded data packets and $\mathbf{G} = [\mathbf{g}_1\mathbf{g}_2 \dots \mathbf{g}_{g+r}]$ that contains $g+r$ randomly generated encoding vectors, where r is redundant packets. In order for a sink to successfully recreate the original data packets, it must receive g linear independent coded packets and encoding vectors. All received coded packets are placed in the matrix $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1\hat{\mathbf{x}}_2 \dots \hat{\mathbf{x}}_g]$ and all encoding vectors are placed in the matrix $\hat{\mathbf{G}} = [\hat{\mathbf{g}}_1\hat{\mathbf{g}}_2 \dots \hat{\mathbf{g}}_g]$. The original data $\mathbf{M}$ can then be decoded as $\hat{\mathbf{M}} = \hat{\mathbf{X}} \times \hat{\mathbf{G}}^{-1}$.

All operation are performed over a Galois field of size q. Thus the probability that the source generates a linear dependant combination becomes an important factor. This probability depends on q and $g' = g - \tilde{g}$, where $\tilde{g}$ is the number of linear independent packets received by the sink and g' thus is the number of needed packets at the sink. The following bound for linear independence is assumed in an alternative form in [11], [12] and is said to hold when q is high .

$$P \leq 1 - \frac{1}{q^{g'}} \tag{4}$$

We provide the following intuitive interpretation, where $\tilde{\mathbf{G}}$ is a matrix of dimension $\tilde{g} \times g$ that contains all received linear independent encoding vectors at the sink. The problem of drawing an encoding vector $\mathbf{g}_i$ that is linear independent of all other $\tilde{g}$ linear independent rows in $\tilde{\mathbf{G}}$ is equal to drawing a linear independent combination of length g'. This is because the degrees of freedoms of any $\mathbf{g}_i$ can be reduced to at most g', alternatively at most g' indices of any $\mathbf{g}_i$ are non zero when all pivot elements in $\tilde{\mathbf{G}}$ is subtracted from $\mathbf{g}_i$. For the remaining sequence of length g' to be dependent it must consist of all zeros which have the probability $\frac{1}{q^{g'}}$.

However we consider cases where q is low and thus we need to estimate to what extend the bound holds for low values of q. To achieve this we have generated a large number, 100.000, square matrices of dimension g consisting of Galois elements with $GF(2)$ and $GF(2^8)$ and tested how many of these was linear independent. For given q and g the probability that a generated combination is linear dependent can be written as:

$$1 - \prod_{i=1}^{g} \left(1 - \frac{1}{q^i}\right) \tag{5}$$

Although the results in Table Ib does not necessarily guarantee equality the empirically obtained values is very close to the calculated and therefore this approximation, if any, is acceptable in the following analysis.

g	calculated	empirical
2	62.5	62.6
4	69.2	69.3
8	71.0	70.9
16	71.1	71.2
32	71.1	71.0
64	71.1	71.1

(a) $GF(2)$

g	calculated	empirical
2	3.92E-3	3.99E-3
4	3.92E-3	3.67E-3
8	3.92E-3	3.87E-3
16	3.92E-3	3.78E-3
32	3.92E-3	3.84E-3
64	3.92E-3	3.88E-3

(b) $GF(2^8)$

TABLE I: Calculated and empirical determined probability of generating a linear dependent matrix of size g.

The state machine of a single sink receiving g packets has *g+1* states depicted in Figure 3.

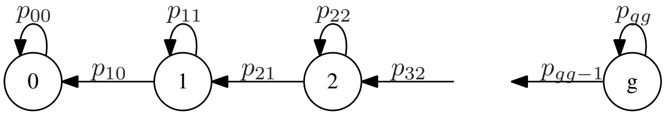


Fig. 3: Markov Chain for a single node that receives g packets.

The probability that a packet is useful at a sink is the probability it is received multiplied with the probability that it is independent

$$P_{i \to (i-1)} = (1-p)\left(1 - \frac{1}{q^i}\right)$$

The probability that it is not useful is therefore the probability that it was not received plus the probability that it was received but was dependent.

$$\begin{aligned} P_{i \to i} =& 1 - (1-p)\left(1 - \frac{1}{q^i}\right) = 1 - 1 + \frac{1}{q^i} + p - p\frac{1}{q^i} \\ =& p + (1-p)\frac{1}{q^i} \end{aligned}$$

These probabilities form the transition matrix $\mathbf{C}$.

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ (1-p)(1-\frac{1}{q^1}) & p+(1-p)\frac{1}{q^1} & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & (1-p)(1-\frac{1}{q^g}) & p+(1-p)\frac{1}{q^g} \end{bmatrix}$$

TABLE II: Transmission matrix for coding.

$$\mathbf{Q}^i = \mathbf{S} \times \mathbf{C}^i \tag{6}$$

Thus the expected number of transmissions for one packet of the total g packets can be found by:

$$E[t_x] = \frac{1}{g}\sum_{i=0}^{\infty} 1 - \left(\mathbf{Q}^i_{(g,0)}\right)^N \tag{7}$$

D. Systematic Network Coding

Systematic RLNC consists of two phases. In the first phase all packets g in the generation are broadcasted uncoded. In the second phase coded packets are broadcasted and thus each node can be in $g+1$ one states, where state zero indicates that no packets are missing and state g that all packets in the generation are missing. Uncoded packets can be perceived as coded packets with a trivial encoding vector where a single element in the encoding vector $\mathbf{g}_i$ is one and all other, $g-1$, elements is zero. Thus we can generate an uncoded packet $\mathbf{y}$ from its trivial encoding vector $\mathbf{h}$, $\mathbf{y} = \mathbf{M} \times \mathbf{h}$. In this way we can construct $\mathbf{Y} = [\mathbf{y}_1\mathbf{y}_2 \ldots \mathbf{y}_g]$ that consists of g uncoded data packets and $\mathbf{H} = [\mathbf{h}_1\mathbf{h}_2 \ldots \mathbf{h}_g]$ that contains the g independent trivial encoding vectors. Furthermore we construct $\mathbf{X} = [\mathbf{x}_1\mathbf{x}_2 \ldots \mathbf{x}_r]$ that consists of r coded data packet and $\mathbf{G} = [\mathbf{g}_1\mathbf{g}_2 \ldots \mathbf{g}_r]$ that contains r randomly generated encoding vectors. For a sink to successfully recreate the original data packets, it must receive g linear independent packets and encoding vectors. Thus g received uncoded and coded packets are placed in the matrix $[\hat{\mathbf{Y}}\hat{\mathbf{X}}] = [\hat{\mathbf{y}}_1\hat{\mathbf{y}}_2 \ldots \hat{\mathbf{x}}_{(g-i)}\ \hat{\mathbf{x}}_1\hat{\mathbf{x}}_2 \ldots \hat{\mathbf{x}}_i]$ and the g corresponding encoding vectors are placed in the matrix $[\hat{\mathbf{H}}\hat{\mathbf{G}}] = [\hat{\mathbf{h}}_1\hat{\mathbf{h}}_2 \ldots \hat{\mathbf{h}}_{(g-i)}\ \hat{\mathbf{g}}_1\hat{\mathbf{g}}_2 \ldots \hat{\mathbf{g}}_i]$. The original data $\mathbf{M}$ can then be decoded as $\hat{\mathbf{M}} = [\hat{\mathbf{Y}}\hat{\mathbf{X}}] \times [\hat{\mathbf{H}}\hat{\mathbf{G}}]^{-1}$.

In phase one the transition matrix for a single sink is identical to that of broadcast, however here we consider the transmission of g packets instead of one packet, thus the transition matrix have $g+1$ states:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ (1-p) & p & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & (1-p) & p \end{bmatrix}$$

TABLE III: Transmission matrix for broadcasting.

The probability distribution of the first phase is the input to the transition matrix of the second phase and thus the probabilities in any of the two phases can be calculated as:

$$\mathbf{R}^i = \begin{cases} \mathbf{S} \times \mathbf{T}^i & \text{for } 0 \le i \le g \\ \mathbf{S} \times \mathbf{T}^g \times \mathbf{C}^i & \text{for } g < i \end{cases} \tag{8}$$

Thus the expected number of transmissions can be found by:

$$E[t_x] = \frac{1}{g} \sum_{i=0}^{\infty} 1 - \left(\mathbf{R}^i_{(g,0)}\right)^N \tag{9}$$

We note that the performance of systematic network coding is equal or better than that of pure network coding. For the trivial coded packets in systematic coding the probability of linear dependencies is zero, for pure network coding the probability is non-zero but very small, see Equation 5. For the non-trivially coded packets the probability of linear independence is identically for systematic and pure network coding.

E. Results

Here we compare the performance of RLNC with unicast and broadcast. We assume *p=0.3* as we have previously observed such p [13]. In Figure 4 the performance of unicast, broadcast and network coding at different settings is plotted for an increasing number of sinks.

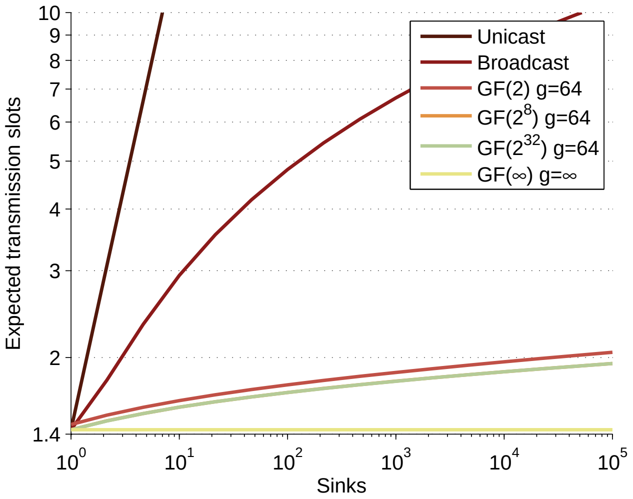


Fig. 4: Expected number of transmission per packet, $p = 0.3$.

Unicast is not designed to perform this type of transmission and therefore it is not surprising that it performs the worst. Broadcast performs better, however, as the number of sinks increases it suffers from the fact that all sinks must receive *all* original packets and thus retransmissions of packets become necessary. For RLNC this is not the case, instead all sinks only have to receive a number of *any* independent coded packets and can then decode the original data. When g is fixed RLNC with GF(2) performs the worst of the RLNC approaches and GF(2^8) and GF(2^{32}) performs the same, therefore we do not consider GF(2^{32}) in the following.

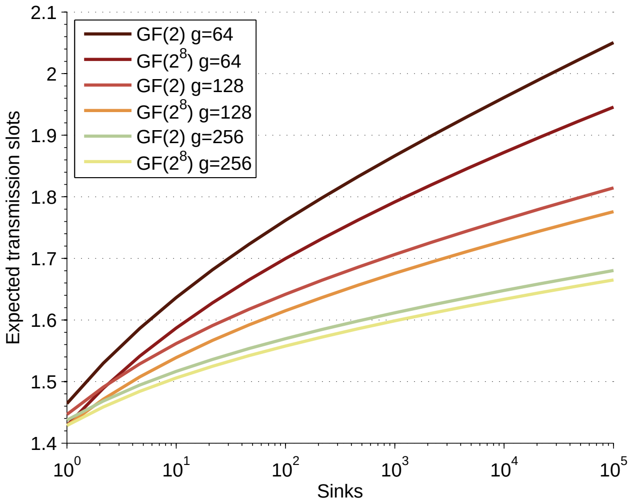


Fig. 5: Expected number of transmission per packet, $p = 0.3$.

In Figure 5 we see that network performance improves when either the field or generation size is increased. When the field size increases the probability of linear dependency decreases and when the generation size increase the uncertainty of how many packets will be lost at each sink decreases. Generally the benefit of doubling g is greater than going from GF(2) to GF(2^8) and additionally the performance benefit of moving from GF(2) to GF(2^8) decreases as g increases. The reason is that for high values of g linear dependency becomes less important because only the very last packet will have a low probability of being linear independent. Thus when g is increased the ratio of packet that have low probability of being linear independent decreases.

Thus if the throughput that can be achieved with GF(2) allows for a substantially higher value of g compared to an implementation with GF(2^8), it may be beneficial to use the small field in some cases.

IV. Implementation

To get a feel for the achievable encoding and decoding throughput we implemented RLNC based on GF(2). In GF(2) adding two packets simplifies to the XOR operation. Encoding a packet in GF(2) can be performed in two simple steps. First the encoding vector is generated as a random bit vector, where the indices in the vector corresponds to packets in the original data set i.e. index one corresponds to packet one. The second step is performed by iterating over the encoding vector and adding packets where the corresponding index in the encoding vector is 1. The following listing shows the encoding algorithm in pseudo code, where $\mathbf{M}$ is the data buffer containing all original packets, $\mathbf{g}$ is an encoding vector and $\mathbf{x}$ is the resulting encoded packet.

```
procedure EncodePacket($\mathbf{M}$,$\mathbf{x}$,$\mathbf{g}$)
    $\mathbf{x} = \bar{0}$
    for each bit $b$ in $\mathbf{g}$ do
        if $b$ equal 1 then
            $i$ = position of $b$ in $\mathbf{g}$
            $\mathbf{x}$ = XOR($\mathbf{x}$, $\mathbf{M}[i]$)
        end if
    end for
end procedure
```

Decoding is performed on the run in two steps with a slightly modified Gauss-Jordan algorithm. Note that this approach is different from what is typically done in implementations for higher field sizes where the encoding matrix is inverted and then subsequently multiplied with the data matrix. Thus the received data at the sink is always decoded as much as possible and the load on the CPU is distributed evenly. In the first step we reduce the incoming encoded packet by performing a forward substitution of already received packets. This is done by inspecting the elements of the encoding vector from start to end and thus determining which original packets the coded packet is a combination of. If an element is 1 and we have already identified a packet with this element as a pivot element we subtract that packet from the coded packet and continue the inspection. If an element is 1 and we have not already identified a packet where this element is a pivot element we have identified a pivot packet and continue to the second stage of the decoding. Note that if we are able to subtract all information contained in the received encoded packet, it will contain no information useful to us and can be discarded.

In the second step we perform backward substitution with the newly identified pivot packet. We do this by subtracting the pivot packet from previously received packets for which the corresponding encoding vector indicates that the particular packet is a combination of the pivot packet. The following listing shows the decoding algorithm in pseudo code, where $\hat{\mathbf{M}}$ is the packet decode buffer of packets received and decoded so far and $\hat{\mathbf{G}}$ is the corresponding encoding vector buffer, $\hat{\mathbf{x}}$ is a newly received encoded packet and $\hat{\mathbf{g}}$ is the newly received encoding vector.

```
procedure DecodePacket($\hat{\mathbf{M}}$,$\hat{\mathbf{G}}$,$\hat{\mathbf{x}}$,$\hat{\mathbf{g}}$)
    pivotposition = 0
    pivotfound = false
    for each bit $b$ in $\hat{\mathbf{g}}$ do                ▷ Forward Substitution
        if $b$ equal 1 then
            $i$ = position of $b$ in $\hat{\mathbf{g}}$
            if $i$'th packet is in $\hat{\mathbf{M}}$ then
                $\hat{\mathbf{g}}$ = XOR($\hat{\mathbf{g}}$,$\hat{\mathbf{G}}[i]$)
                $\hat{\mathbf{x}}$ = XOR($\hat{\mathbf{x}}$,$\hat{\mathbf{M}}[i]$)
            else
                pivotfound = true
                pivotposition = $i$
            end if
        end if
    end for
    if pivotfound equal false then
        Exit procedure ▷ The packet was linear dependant
    end if
    for each packet $j$ in $\hat{\mathbf{M}}$ do     ▷ Backward Substitution
        $k$ = $\hat{\mathbf{G}}[j]$
        if bit at pivotposition in $k$ equal 1 then
            $\hat{\mathbf{G}}[j]$ = XOR($\hat{\mathbf{G}}[j]$,$\hat{\mathbf{g}}$)
            $\hat{\mathbf{M}}[j]$ = XOR($\hat{\mathbf{M}}[j]$,$\hat{\mathbf{x}}$)
        end if
    end for
    $\hat{\mathbf{G}}$[pivotposition] = $\hat{\mathbf{g}}$
    $\hat{\mathbf{M}}$[pivotposition] = $\hat{\mathbf{x}}$
end procedure
```

The algorithm can also be used unmodified in a systematic coding approach, in which case we only have to ensure that uncoded packets are treated as pivot packets. Based on these algorithms we have implemented a coding library designed to deliver high throughput and optimized through assembly and SIMD instructions. Subsequently the implementation was ported to Symbian to allow for tests on a mobile platform. All implementations are single threaded. We used the following platforms for the tests.

1) Nokia N95-8GB, ARM 11 332 MHz CPU, 128 MB ram, Symbian OS 9.2.

2) Lenovo T61p, 2.53 GHz Intel Core2Duo, 2 GB ram, Kubuntu 8.10 64bit.

We tested the performance of the implementation by encoding and decoding without transmission over any network at different generation sizes from 16 to 256. First a large file, 5 MB for the phone and 128 MB for the PC, was divided into packets of 1500 Bytes. These packets were then split into generations of size equal to the specified generation size. From each of these generations packets were generated and saved. Then packets from each generation were read and decoded until the original data was recreated. Encoding and decoding time were measured and from this the throughput was calculated.

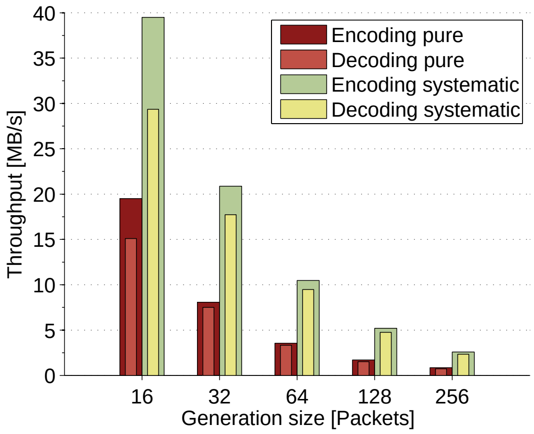


Fig. 6: Encoding and Decoding throughput on a mobile phone.

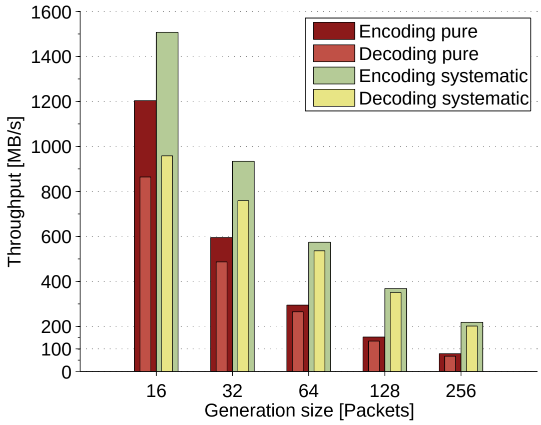


Fig. 7: Encoding and Decoding throughput on a laptop.

Both encoding and decoding throughput is considerably higher than any other reported result known to the authors and the throughput is approximately a first order function of the generation size. Note that for the systematic approach 70% of the packets where uncoded and 30% coded. In a real wireless scenario the ratio of coded vs. uncoded packets depends on the error probability of the link. In the pure approach all packets are coded, thus the throughput for the pure approach is equal to the worst case throughput of the systematic approach.

V. Conclusion

In this paper we have proposed to base RLNC on the binary Galois field in order to decrease the computational complexity. Additionally we have proposed techniques for reducing the amount of coding needed, which can help to increase throughput and decrease energy consumption of NC implementations. The proposed approach have been analyzed from a network point of view. The approach have been implemented and we have demonstrated that high encoding and decoding throughputs can be achieved on both mobile phones and laptops.

VI. Acknowledgement

We would like to thank Ralf Koetter for an interesting discussion in the Biergarten and Muriel Medard for the discussion on network coding and cooperative wireless networks. Furtheremore, we would like to thank Nokia for providing technical support and mobile phones. Special thanks to Mika Kuulusa, Gerard Bosch, Harri Pennanen, Nina Tammelin, and Per Moeller from Nokia. This work was partially financed by the X3MP project granted by Danish Ministry of Science, Technology and Innovation.

References

[1] "The network coding home page." https://hermes.lnt.e-technik.tu-muenchen.de/DokuWiki/doku.php?id=network_coding:bibliography_for_network_coding. Extensive (250+) but incomplete list of publications related to Network Coding.

[2] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, "Cautious view on network coding - from theory to practice," *Journal of Communications and Networks (JCN)*, 2008.

[3] J.-S. Park, M. Gerla, D. S. Lun, Y. Yi, and M. Medard, "Codecast: a network-coding-based ad hoc multicast protocol," *Wireless Communications, IEEE [see also IEEE Personal Communications]*, vol. 13, no. 5, pp. 76–81, October 2006.

[4] D. Nguyen, T. Nguyen, and B. Bose, "Wireless broadcasting using network coding," in *Third Workshop on Network Coding, Theory, and Applications*, January 2007.

[5] L. Popova, A. Schmidt, W. Gerstacker, and W. Koch, "Network coding assisted mobile-to-mobile file transfer," in *Australasian Telecommunication Networks and Applications Conference (ATNAC)*, December 2007.

[6] M. Wang and B. Li, "How practical is network coding?," *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pp. 274–278, June 2006.

[7] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *INFOCOM*, pp. 1082–1090, 2007.

[8] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 169–180, 2007.

[9] H. Shojania and B. Li, "Parallelized progressive network coding with hardware acceleration," *Quality of Service, 2007 Fifteenth IEEE International Workshop on*, pp. 47–55, June 2007.

[10] M. Xiao, T. Aulin, and M. Médard, "Systematic binary deterministic rateless codes," in *Proceedings IEEE International Symposium on Information Theory*, 2008.

[11] D. E. Lucani, M. Stojanovic, and M. Mdard, "Random linear network coding for time division duplexing: When to stop talking and start listening," *CoRR*, vol. abs/0809.2350, 2008. informal publication.

[12] 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.

[13] J. Heide, M. V. Pedersen, F. H. Fitzek, T. V. Kozlova, and T. Larsen, "Know your neighbour: Packet loss correlation in ieee 802.11b/g multicast," in *The 4th International Mobile Multimedia Communications Conference (MobiMedia '08)*, (Oulu, Finland), July 7-9 2008.

Paper 2

Reducing Computational Overhead of Network Coding with Intrinsic Information Conveying

Janus Heide
Qi Zhang
Morten V. Pedersen
Frank H.P. Fitzek

IEEE Vehicular Technology Conference (VTC)
Transmission Technologies Track
San Francisco, CA, USA, September 5–8, 2011

Reducing Computational Overhead of Network Coding with Intrinsic Information Conveying

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.

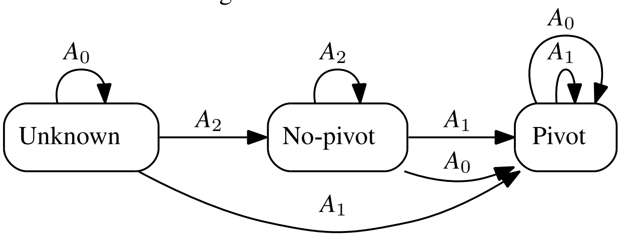


Fig. 1: The three states and the possible transitions.

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 $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1 \hat{\boldsymbol{x}}_2 \ldots \hat{\boldsymbol{x}}_{g'}]$ and all coding vectors are in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1 \hat{\boldsymbol{g}}_2 \cdots \hat{\boldsymbol{g}}_{g'}]$. 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.

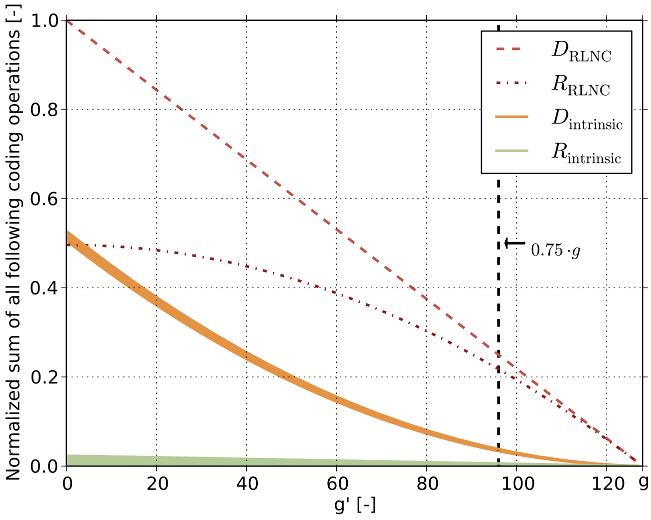


Fig. 2: Number of operations needed to finish decoding as a function of g'.

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.

Paper 3

Decoding Algorithms for Random Linear Network Codes

Janus Heide
Morten V. Pedersen
Frank H.P. Fitzek

IFIP International Conferences on Networking
Workshop on Network Coding Applications and Protocols (NC-Pro)
Vol. 6827. Lecture Notes in Computer Science. pp. 129–137
Valencia, Spain, May 12, 2011

Decoding Algorithms for Random Linear Network Codes

Janus Heide, Morten V. Pedersen, and Frank H.P. Fitzek

Falculty of Engineering and Science
Aalborg University, Aalborg, Denmark
jah@es.aau.dk

Abstract. We consider the problem of efficient decoding of a random linear code over a finite field. In particular we are interested in the case where the code is random, relatively sparse, and use the binary finite field as an example. The goal is to decode the data using fewer operations to potentially achieve a high coding throughput, and reduce energy consumption. We use an on-the-fly version of the Gauss-Jordan algorithm as a baseline, and provide several simple improvements to reduce the number of operations needed to perform decoding. Our tests show that the improvements can reduce the number of operations used during decoding with 10-20% on average depending on the code parameters.

Keywords: Network Coding; Algorithms; Implementation

1 Introduction

When implementing and deploying Network Coding (NC) at least two performance criteria are important; the magnitude of overhead added by the code, and the speed at which encoding, recoding and decoding can be performed. We consider the last issue and note that it is trivial to implement encoding such that the minimal number of operations are used. As recoding is similar to encoding we turn our attention to the problem of fast decoding.

A popular approach to network coding is Random Linear Network Coding (RLNC), introduced in [2]. It is based on finite fields, and it has been shown that high coding throughput can be obtained with this code when the binary finite field is used [1]. Additionally, as a randomly drawn element from the binary field is zero with high probability (50%), the resulting code will be sparse.

As data is encoded and recoded in a random way, there is no special structure or shortcut to exploit when performing decoding. Instead we are left with the tedious task of determining the inverse of operations performed during encoding/recoding. Additionally we prefer to perform decoding as packets arrive in order to avoid a large decoding delay when the final packet arrives. We know the resulting code is sparse, and therefore propose some simple mechanisms to utilize this fact. We have implemented these and their impact on the number of operations used during decoding.

In the remainder of this paper we introduce the used encoding approach, several decoding optimizations, and their measured impact on the decoding.

2 Coding Algorihtms

We consider encoding packets from some data to be sent from a source to a sink, we denote this data the generation. The generation consists of g pieces, called symbols each with a size of m bits, where g is called the generation size, and thus the generation contains $g \cdot m$ bits of data. The g symbols are arranged in the matrix $\boldsymbol{M} = [\boldsymbol{m}_1; \boldsymbol{m}_2; \dots; \boldsymbol{m}_g]$, where $\boldsymbol{m}_i$ is a column vector. In practise some original file or data stream may be split into several generations, but here we only consider a single generation.

To generate a new encoded symbol $\boldsymbol{x}$, $\boldsymbol{M}$ is multiplied with a randomly generated coding vector $\boldsymbol{g}$ of length g, $\boldsymbol{x} = \boldsymbol{M} \times \boldsymbol{g}$. In this way we can construct $g + r$ coded symbols and coding vectors, where r is any number of redundant symbols as the code is rateless. When a coded symbol is transmitted on the network it is accompanied by its coding vector, and together they form a coded packet. A practical interpretation is that each coded symbol, is a combination or mix of the original symbols from the generation. The benefit is that nearly infinite coded symbols can be created.

$$\overbrace{\boxed{\text{Existing protocol header}}\boxed{\text{Coding vector } \boldsymbol{g}}\boxed{\text{Coded symbol } \boldsymbol{x}}}^{\text{Coded packet}}$$

2.1 Decoding

A sink must receive g linearly independent symbols and coding vectors from the generation to decode the data successfully. All received symbols are placed in the matrix $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \dots, \hat{\boldsymbol{x}}_g]$ and all coding vectors are placed in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1, \hat{\boldsymbol{g}}_2, \dots, \hat{\boldsymbol{g}}_g]$, we denote $\hat{\boldsymbol{G}}$ the decoding matrix. Thus the vectors and symbols are row vectors in $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{X}}$ respectively as this is more convenient during recoding. Hence we may perform any row operation on $\hat{\boldsymbol{G}}$ if we perform the same row operation on $\hat{\boldsymbol{X}}$.

The original data $\boldsymbol{M}$ can then be decoded as $\hat{\boldsymbol{M}} = \hat{\boldsymbol{X}} \times \hat{\boldsymbol{G}}^{-1}$ by the decoder. The problem is how to achieve this in an efficient way. We note that row operations on $\hat{\boldsymbol{X}}$ are more computationally expensive compared to operations on $\hat{\boldsymbol{G}}$, as generally $m >> g$.

Elements in the matrices are indexed row-column, thus $\hat{\mathbf{G}}[i, j]$ is the element in $\hat{\mathbf{G}}$ on the intersection between the i'th row and the j'th column. The i'th row in the matrix is indexed as $\hat{\mathbf{G}}[i]$. Initially no packets have been received, thus $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{X}}$ are zero matrices. As we operate in the binary finite field we denote bitwise XOR of two bit strings of the same length as $\oplus$.

```
Algorithm 1: Decoder initial state
  Input: $g, m$
  Data: $\hat{\boldsymbol{G}} \leftarrow \mathbf{0}_{g \times g}$          ▷ The decoding matrix
  Data: $\hat{\boldsymbol{X}} \leftarrow \mathbf{0}_{g \times m}$          ▷ The (partially) decoded data
  Data: rank $\leftarrow 0$                    ▷ the rank of $\hat{\boldsymbol{G}}$
```

2.2 Basic

As a reference we use the *basic* decoder algorithm, see Algorithm 5, described in [1]. This algorithm is a modified version of the Gauss-Jordan algorithm. On each run the algorithm attempts to get the decoding matrix into reduced echelon form. First the received vector and symbol $\hat{\boldsymbol{g}}$ and $\hat{\boldsymbol{x}}$ is forward substituted into the previous received vectors and symbols $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{X}}$ respectively, and subsequently backward substitution is performed. If the packet was a linear combination of previous received packets it is reduced to the zero-vector $\mathbf{0}_g$ and discarded.

Algorithm 2: ForwardSubstitute

```
Input: $\hat{x}$,$\hat{g}$
pivotPosition ← 0                          ▷ 0 Indicates that no pivot was found
for $i \leftarrow 1 : g$ do
    if $\hat{g}[i] = 1$ then
        if $\hat{G}[i, i] = 1$ then
            $\hat{g} \leftarrow \hat{g} \oplus \hat{G}[i]$          ▷ substitute into new vector
            $\hat{x} \leftarrow \hat{x} \oplus \hat{X}[i]$          ▷ substitute into new symbol
        else
            pivotPosition ← $i$                     ▷ pivot element found
            break
return pivotPosition
```

Algorithm 3: BackwardsSubstitute

```
Input: $\hat{x}$,$\hat{g}$, pivotPosition
for $i \leftarrow$ (pivotPosition $-$ 1) : 1 do
    if $\hat{G}[i,$ pivotPosition$] = 1$ then
        $\hat{G}[i] \leftarrow \hat{G}[i] \oplus \hat{g}$          ▷ substitute into old vector
        $\hat{X}[i] \leftarrow \hat{X}[i] \oplus \hat{x}$          ▷ substitute into old symbol
```

Algorithm 4: InsertPacket

```
Input: $\hat{x}$,$\hat{g}$, pivotPosition
$\hat{G}$[pivotposition] = $\hat{g}$
$\hat{X}$[pivotposition] = $\hat{x}$
```

Algorithm 5: DecoderBasic

```
Input: $\hat{x}$,$\hat{g}$
pivotPosition = ForwardSubstitute($\hat{x}$,$\hat{g}$)
if pivotPosition > 0 then
    BackwardsSubstitute($\hat{x}$,$\hat{g}$, pivotPosition)
    InsertPacket($\hat{x}$,$\hat{g}$, pivotPosition)
    rank++
return rank
```

2.3 Suppress Null (SN)

To avoid wasting operations on symbols that does not carry novel information, we record the operations performed on the vector. If the vector is reduced to the zero vector, the packet was linearly dependent and the recorded operations are discarded. Otherwise the packet was novel and the recorded operations are executed on the symbol. This reduces the computational cost when a linear dependent packet is received. This is most likely to occur in the end phase of the decoding, thus it is most beneficial for small generation sizes. In real world scenarios the probability of receiving a linearly dependent packet can be high, in which cases this approach would be beneficial. To implement this, line 1 in Algorithm 5 is replaced with Algorithm 7.

Algorithm 6: ExecuteRecipe

```
Input: $\hat{\boldsymbol{x}}$,recipe
for $i \leftarrow 1 : g$ do
    if recipe[$i$] $= 1$ then
        $\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} \oplus \hat{\boldsymbol{X}}[i]$        ▷ substitute into symbol
```

Algorithm 7: ForwardSubstituteSuppressNull

```
Input: $\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{g}}$
pivotPosition ← 0        ▷ 0 Indicates that no pivot was found
recipe ← $\mathbf{0}_g$
for $i \leftarrow 1 : g$ do
    if $\hat{\boldsymbol{g}}[i] = 1$ then
        if $\hat{\boldsymbol{G}}[i, i] = 1$ then
            $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} \oplus \hat{\boldsymbol{G}}[i]$        ▷ substitute into new vector
            recipe[$i$] ← 1
        else
            pivotPosition ← $i$        ▷ pivot element found
            break
if pivotPosition $> 0$ then
    ExecuteRecipe($\hat{\boldsymbol{x}}$,recipe)
return pivotPosition
```

2.4 Density Check (DC)

When forward substitution is performed there is a risk that a high density packets is substituted into a low density packet. The density is defined as Density($\boldsymbol{h}$) $= \frac{\sum_{k=1}^{g}(h_k \neq 0)}{g}$, which is the number of non-zeros in the vector, and where $\boldsymbol{h}$ is the coding vector. Generally a sparse packet requires little work to decode and a dense packet requires much work to decode. When a vector is substituted into a sparse vector, the resulting vector will with high probability have higher density and thus *fill-in* occur. To reduce this problem incoming packets can be sorted based on density during forward substitution. When it is detected that two vectors have the same pivot element their densities are compared. The vector with the lowest density is inserted into the decoding matrix. The low density packet is then substituted into the high density packet, and the forward substitution is continued with the resulting packet. To implement this, line 1 in Algorithm 5 is replaced with Algorithm 8.

Algorithm 8: ForwardSubstituteDensityCheck

```
Input: $\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{g}}$
pivotPosition $\leftarrow 0$                    ▷ 0 Indicates that no pivot was found
for $i \leftarrow 1 : g$ do
    if $\hat{\boldsymbol{g}}[i] = 1$ then
        if $\hat{\boldsymbol{G}}[i, i] = 1$ then
            if Density($\hat{\boldsymbol{g}}$) $<$ Density($\hat{\boldsymbol{G}}[i]$) then
                $\hat{\boldsymbol{g}} \leftrightarrow \hat{\boldsymbol{G}}[i]$              ▷ swap new vector with old vector
                $\hat{\boldsymbol{x}} \leftrightarrow \hat{\boldsymbol{X}}[i]$              ▷ swap new symbol with old symbol
            $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} \oplus \hat{\boldsymbol{G}}[i]$
            $\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} \oplus \hat{\boldsymbol{X}}[i]$
        else
            pivotPosition $\leftarrow i$                     ▷ pivot element found
            break
return pivotPosition
```

2.5 Delayed Backwards Substitution (DBS)

To reduce the *fill-in* effect the backwards substitution is postponed until the decoding matrix has full rank. Additionally it is not necessary to perform any backwards substitution on the vectors because backwards substitution is performed starting from the last row. Hence when backwards substitution of a packet is complete, that packet has a pivot element for which all other encoding vectors are zero. This approach is only semi on-the-fly, as only some decoding is performed when packets arrive. Therefore the decoding delay when the final packet arrive will increase. This is implemented with Algorithm 9

Algorithm 9: DecoderDelayedBackwardsSubstitution

```
Input: $\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{g}}$
pivotPosition = ForwardSubstitute($\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{g}}$)
if pivotPosition $> 0$ then
    InsertPacket($\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{g}}$, pivotPosition)
    rank++
if rank $= g$ then
    BackwardsSubstituteFinal()
return rank
```

Algorithm 10: BackwardsSubstituteFinal

```
for $i \leftarrow g : 2$ do
    for $j \leftarrow (i-1) : 1$ do                              ▷ All rows above
        if $\hat{\boldsymbol{G}}[j,i] = 1$ then
            $\hat{\boldsymbol{X}}[j] \leftarrow \hat{\boldsymbol{X}}[j] \oplus \hat{\boldsymbol{X}}[i]$        ▷ substitute into the symbol
```

2.6 Density Check, and Delayed Backwards Substitution (DC-DBS)

When DC and DBS are combined vectors are sorted so sparse vectors are kept at the top of the decoding matrix while dense vectors are pushed downwards. Because backwards substitution is performed only when the rank is full, no fill-in occurs during backwards substitution, as only fully decoded packets are substituted back. To implement this, line 1 in Algorithm 9 is replaced with Algorithm 8.

2.7 Suppress Null, Density Check, and Delayed Backwards Substitution (SN-DC-DBS)

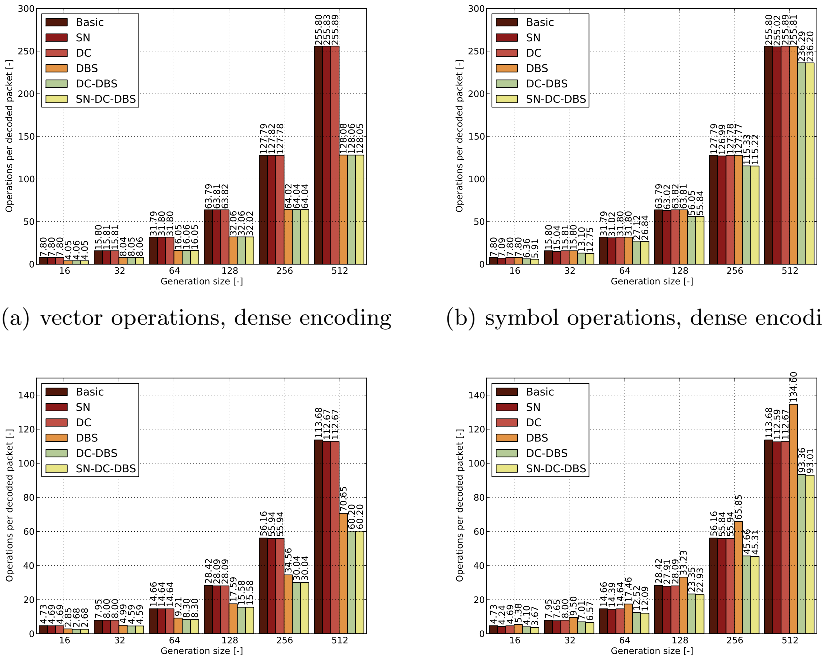
To reduce the cost of receiving linear dependent packets we include SN, by replacing line 1 in Algorithm 9 with Algorithm 11.

Algorithm 11: ForwardSubstitute-SN-DC

```
Input: $\hat{\boldsymbol{x}}$,$\hat{\boldsymbol{g}}$
pivotPosition $\leftarrow 0$                        ▷ 0 Indicates that no pivot was found
recipe $\leftarrow \boldsymbol{0}_g$
for $i \leftarrow 1 : g$ do
    if $\hat{\boldsymbol{g}}[i] = 1$ then
        if $\hat{\boldsymbol{G}}[i,i] = 1$ then
            if Density($\hat{\boldsymbol{g}}$) < Density($\hat{\boldsymbol{G}}[i]$) then
                ExecuteRecipe($\hat{\boldsymbol{x}}$,recipe)
                recipe $\leftarrow \boldsymbol{0}_g$                              ▷ reset recipe
                $\hat{\boldsymbol{g}} \leftrightarrow \hat{\boldsymbol{G}}[i]$                ▷ swap new vector with old vector
                $\hat{\boldsymbol{x}} \leftrightarrow \hat{\boldsymbol{X}}[i]$                ▷ swap new symbol with old symbol
            $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} \oplus \hat{\boldsymbol{G}}[i]$                ▷ substitute into new vector
            recipe$[i] \leftarrow 1$
    else
        pivotPosition $\leftarrow i$                        ▷ pivot element found
        break
if pivotPosition $> 0$ then
    ExecuteRecipe($\hat{\boldsymbol{x}}$,recipe)
return pivotPosition
```

3 Results

We have decoded a large number of generations with each of the optimizations, and measured the used vector and symbol operations which is $\oplus$ of two vectors, and two symbols respectively. We have considered two densities while encoding, $d = \frac{1}{2}$ and $d = \frac{\log_2(g)}{g}$ which we denote dense and sparse respectively. $d = \frac{1}{2}$ gives the lowest probability of linear dependence, and $d = \frac{\log_2(g)}{g}$ is a good trade-off between linear dependence and density. As a reference the mean number of both vector and symbol operations during encoding of one packet can be calculated as $\frac{g}{2}$ and $\log_2(g)$ for the dense and sparse case respectively.



(a) vector operations, dense encoding

(b) symbol operations, dense encoding

(c) vector operations, sparse encoding

(d) symbol operations, sparse encoding

Fig. 1. Vector and symbol operations during decoding, when $d = \frac{1}{2}$ and $d = \frac{\log_2(g)}{g}$.

With the *Basic* algorithm the number of operations during encoding and decoding is identical for the dense case, see Fig 1(a) and 1(b) and calculate $\frac{g}{2}$. For the sparse case the number of operations is significantly higher for decoding than for encoding, see Fig 1(c) and 1(d), and calculate $\log_2(g)$.

When *Suppress Null* is used the number of reduced symbol operations can be observed by subtracting the number of symbol operations from the number of vector operations. The reduction is highest for small g, which is where the ratio of linearly dependent packet is largest. For $g = 16$ and $g = 32$ the reduction in symbol operations is 9.5% and 4.4% respectively, for high $g's$ the reduction is approximately 0%.

Density Check reduces the number of operations for the sparse case marginally, but has no effect for the dense case.

The *Delayed Backwards Substitution* decreases the number of vector operations with approximately 40% when the encoding is sparse, and 50% when the encoding is dense as no operations needs to be performed on the vectors during backwards substitution. Interestingly the number of symbol operations increase significantly for the sparse encoding.

In *DC-DBS*, *density check* and *delayed backwards substitution* are combined, and the number of both data and vector operations are significantly reduced. For the sparse case the reduction is approximately 50% of the vector operations, and almost 20% of the symbol operations. For the dense case the reduction is approximately 50% of the vector operations, and almost 10% of the symbol operations. Interestingly the number of vector and symbol operations is lower for decoding compared to encoding, in the dense case. Hence the combination of *DC* and *DBS* is significantly better than the two alone, as the expensive symbol operations are reduced.

With *SN-DC-DBS* we additionally include *Suppress Null*, the number of symbol operations is reduced slightly for high g and significantly for low g.

4 Conclusion

The considered decoding optimizations have been shown to reduce the number of necessary operations during decoding. The reduction in symbol decoding operations is approximately 10%, and 20%, when the density during encoding is dense and sparse respectively. In both cases the number of vector operations is approximately halved. Surprisingly decoding can in some cases be performed with fewer operations than encoding.

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, and the ENOC project in collaboration with Nokia, Oulu.

References

1. Heide, J., Pedersen, M.V., Fitzek, F.H., Larsen, T.: Network coding for mobile devices - systematic binary random rateless codes. In: The IEEE International Conference on Communications (ICC). Dresden, Germany (14-18 June 2009)
2. Ho, T., Koetter, R., Médard, M., Karger, D., ros, M.: 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)

Paper 4

On Code Parameters and Coding Vector Representation for Practical RLNC

Janus Heide
Morten V. Pedersen
Frank H.P. Fitzek
Muriel Médard

IEEE International Conference on Communications (ICC)
Communication Theory Symposium
Kyoto, Japan, June 5–9, 2011

On Code Parameters and Coding Vector Representation for Practical RLNC

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

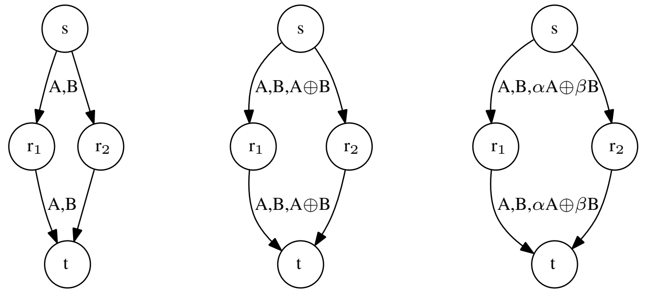
Muriel Médard
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
Email: medard@mit.edu

***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 encoding vector. Increasing the field size and generation size presents a fundamental trade-off between packet-based throughput and operational overhead. On the one hand, decreasing the probability of transmitting redundant packets is beneficial for throughput and, consequently, reduces transmission energy. On the other hand, the decoding complexity and amount of header overhead increase with field size and generation length, leading to higher energy consumption. Therefore, the optimal trade-off is system and topology dependent, as it depends on the cost in energy of performing coding operations versus transmitting data. We show that moderate field sizes are the correct choice when trade-offs are considered. The results show that sparse binary codes perform the best, unless the generation size is very low.**

I. Introduction

Network Coding (NC) is a promising paradigm that breaks with the existing store-and-forward paradigm in computer networks [1]. NC enables coding on the fly at the individual node in the communication network, and thus is fundamentally different from the end-to-end approach of channel and source coding. Thus packets are no longer treated as atomic entities as the number of incoming and outgoing packets per node, is not necessarily equal 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 fits perfectly with the ideas of cooperative and distributed networks.

A promising popular approach, introduced in [2], is RLNC. In RLNC coding is performed at random which minimizes the need for signaling, compared to deterministic codes. Because coding is performed randomly there is a non-zero probability, that a received coded symbol is linearly dependent on already received symbols, and thus unusable. Figure 1 illustrates the benefits of coding. When no coding is used nodes can only forward symbols, and as the relays must forward two different packet for the sink to decode. If binary coding is used an additional symbol can be created, A$\oplus$B, and thus the probability that the relays forward two different symbols increases. If coding is performed over a higher field many more symbols can be created, αA$\oplus\beta$B, and the probability that the relays forward two different symbols increases.



(a) No coding, $P_{\text{fail}} = 50\%$

(b) Binary coding, $P_{\text{fail}} = 33\%$

(c) Higher field coding, $P_{\text{fail}} \approx 1/q$

Fig. 1: Example network with and without coding, s is the source, t is the sink, r_1 and r_2 are relays.

This probability result in the *linear dependence* overhead. The parameters; generation size, field size and density influence this overhead, and are often assumed to be high as this decreases the probability of linear dependence. To decode a received symbol a sink needs the coding vector of the symbol, which describes the coding operation performed during encoding. This information must be included as header information when the coded symbol is transmitted, which results in an additional *header* overhead.

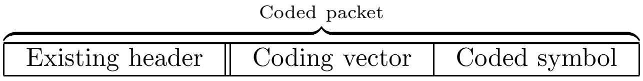
Unfortunately, when coding is performed on a computational device, the parameters also affect the coding throughput, which is the rate at which coding is performed [Mb/s]. Higher values generally result in lower coding throughput [3]–[6]. However, the coding throughput also depends on less deterministic parameters e.g. the hardware platform, programming language, and implementation optimizations. Recently [7] have shown that a systematic code with a RLNC-based redundancy can achieve low computational complexity while remaining binary, but only over a single-hop system.

The objective of this work is to increase the coding throughput without significantly increasing the linear dependence overhead. Our contribution is two fold. In Section II we analyze the impact of changing the field size, generation size, and density, and provide bounds for the resulting linear dependence overhead. In Section III we consider the representation of the coding vector, the resulting header overhead.

II. Coding

The data, of size B that is to be transferred from a source to one or more sinks is divided into generations of size $g \cdot m$, a generation is sometimes also referred to as a source block or a batch. Each generation constitutes g symbols of size m, where g is called the generation size. The g original symbols of length m in one generation, 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. In an application the block of data can be a file or a part of a media stream, and is divided into $\lceil \frac{B}{m} \rceil$ pieces, called symbols. Generation number 0 constitutes the first g symbols, or the first $g \cdot m$ bytes of data, there are $\lceil \frac{B}{g \cdot m} \rceil$ such generations.

To encode a new symbol $\boldsymbol{x}$ from a generation at the source, $\boldsymbol{M}$ is multiplied with a randomly generated coding vector $\boldsymbol{g}$ of length g, $\boldsymbol{x} = \boldsymbol{M} \times \boldsymbol{g}$. In this way we can construct $g + r$ coded symbols and coding vectors, where r is any number of redundant symbols as the code is rateless. When a coded symbol is transmitted on the network it is accompanied by its coding vector, and together they form a coded packet. A practical interpretation is that each coded symbol, is a combination or mix of the original symbols from one generation. The benefit is that nearly infinite coded symbols can be created.



In order for a sink to successfully decode a generation, it must receive g linearly independent symbols and coding vectors from that generation. All received symbols are placed in the matrix $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1; \hat{\boldsymbol{x}}_2; \ldots; \hat{\boldsymbol{x}}_g]$ and all coding vectors are placed in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1; \hat{\boldsymbol{g}}_2; \ldots; \hat{\boldsymbol{g}}_g]$, we denote $\hat{\boldsymbol{G}}$ the decoding matrix. The original data $\boldsymbol{M}$ can then be decoded as $\hat{\boldsymbol{M}} = \hat{\boldsymbol{X}} \times \hat{\boldsymbol{G}}^{-1}$. To spread the computational load this can be performed with an on-the-fly version of Gaussian elimination. In practice if approximately any g symbols from a generation are received the original data in that generation can be decoded. This is a much looser condition, compared to when no coding is used, where exactly all g unique original symbols must be collected [8].

Any node that have received g', where $g' = [2, g]$ is the number of received linearly independent symbols from a generation and is equal to the rank of $\hat{\boldsymbol{G}}$, can recode. All received symbols are placed in the matrix $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1; \hat{\boldsymbol{x}}_2; \ldots; \hat{\boldsymbol{x}}_{g'}]$ and all coding vectors in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1; \hat{\boldsymbol{g}}_2; \ldots; \hat{\boldsymbol{g}}_{g'}]$. To recode a symbol these matrices are multiplied with a randomly generated vector $\boldsymbol{h}$ of length g', $\tilde{\boldsymbol{g}} = \hat{\boldsymbol{G}} \times \boldsymbol{h}$, $\tilde{\boldsymbol{x}} = \hat{\boldsymbol{X}} \times \boldsymbol{h}$. In this way we can construct r' randomly generated recoding vectors and r' recoded symbols. $r' > g'$ is possible, however a node can never create more than g' independent symbols. Note that $\boldsymbol{h}$ is only used locally and that there is no need to distinguish between coded and recoded symbols. In practice this means that a node that have received more than one symbol can recombine those symbols into recoded symbols, similar to the way coded symbols are constructed at the source.

A. Generation Size

The generation size g is the number of symbols over which encoding is performed, and defines the maximal number of symbols that can be combined into a coded symbol. Data is decoded on a per generation level, thus at least g symbols must be received before decoding is possible. Hence the size of a generation $g \cdot m$ dictates the decoding delay which is the minimum amount of data that must be received before decoding is possible.

From a *linear dependence* overhead point of view g should be high, especially in multiple-sink broadcast networks, where a low g increases the amount of expected transmissions per symbol, due to erasures [5]. From a practical point of view, decoding delay and coding throughput must also be considered. For bulk downloads the decoding delay is not important. But for streaming services and Voice over Internet Protocol (VoIP) in particular it is critical, and g must be chosen with care. Additionally a high g generally decreases the coding throughput, thus g must be chosen low enough to ensure satisfactory coding throughput on the given platform.

To achieve reliability in a practical system some signaling is necessary for each generation. A simple form could be to acknowledge when each generation is successfully decoded. Thus the benefits of NC in terms of reduced signaling diminish, when the generation size is decreased, as the number of generations necessary to represent some fixed amount of data increases. This overhead is protocol and topology dependent, and therefore outside the scope of this work.

B. Field Size

The field size, q, defines the size of the finite field over which all coding operations are performed, and thus the number of unique field elements. A necessary but insufficient condition for decoding is that all rows have at least one non-zero scalar. This probability can be found from the probability of receiving a symbol where at least one scalar in the coding vector, that corresponds to a symbol for which the decoder has not yet identified a pivot element, is non-zero. The following bound for linear independence, when each scalar in the coding vector is drawn uniformly, is assumed in an alternative form in [9], [10] and is said to hold when q is high.

$$P_{\text{independent}} \leq 1 - \frac{1}{q^{g-g'}} \tag{1}$$

In [5] we observed the probability of generating g symbols that are not all independent, given by Equation (2), is a good approximation even at low values of q.

$$1 - \prod_{g'=0}^{g-1} \left(1 - \frac{1}{q^{g-g'}}\right) \tag{2}$$

Thus as g' goes towards g it becomes increasingly more difficult to receive useful symbols, because the coding vector must be non-zero in at least one of the $g - g'$ corresponding scalars. This yields the following transition probabilities.

$$P_{g' \rightarrow g'} = \frac{1}{q^{g-g'}} \quad P_{g' \rightarrow g'+1} = 1 - \frac{1}{q^{g-g'}}$$

$$\boldsymbol{P} = \begin{bmatrix} \frac{1}{q^g} & 0 & \cdots & 0 \\ (1-\frac{1}{q^g}) & \frac{1}{q^{g-1}} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & (1-\frac{1}{q^1}) & 1 \end{bmatrix}$$

Thus the expected amount of overhead for a generation can be found by evaluating the probability that the rank is not full after k transmissions, $p(g' \neq g)$. Initially no symbols are received and therefore the starting pmf $\boldsymbol{s}$ is, $\boldsymbol{s} = [1, 0, ...]$. When less than g symbols are received, $p(g' = g) = 0$, and hence the overhead can be evaluated as.

$$\alpha \geq \sum_{k=g'}^{\infty} \boldsymbol{p}^k(g' \neq g), \quad \boldsymbol{p}^k = (\mathbf{P}^k \times \boldsymbol{s}) \tag{3}$$

This can be rewritten to the form in Equation (4).

$$\begin{aligned} \alpha(q,g) &\geq \sum_{g'=0}^{g-1} \left(\left(1 - \frac{1}{q^{g-g'}}\right)^{-1} - 1 \right) \\ &= \sum_{g'=0}^{g-1} \left(\frac{1}{q^{g-g'} - 1} \right) \end{aligned} \tag{4}$$

It might be expected that a decreased density would impact this directly. However as decoding progresses the not-decoded remainder of the coding vectors will go towards a uniform drawn distribution, due to the fill-in effect. Therefor a separate contribution to the overhead stems from the density.

C. Density

The ratio of non-zero scalars in a coding vector is often referred to as the *density*. The density of a coding vector $\boldsymbol{h}$ with a generation size g is defined by Equation (5).

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

A necessary but insufficient condition for decoding is that all columns have at least one non-zero scalar. For a generation a receiving node can have $j = [0, g]$ non-zero columns. The probability that a scalar is non-zero in a received symbol is d. Before the transition there are j non-zero columns, after the transition there are j'. Thus the number of possible combinations for the transition is given by $\binom{g-j}{g-j'}$. $j' - j$ columns becomes non-zero with probability d. $g - j'$ columns remain all-zero with probability $1 - d$. Thus the probability of transition from state j to state j', where $j' \geq j$, is.

$$O_{j \rightarrow j'} = d^{j'-j} \cdot (1-d)^{g-j'} \cdot \binom{g-j}{g-j'} \tag{6}$$

$$\boldsymbol{O} = \begin{bmatrix} (1-d)^g & 0 & \cdots & 0 \\ d \cdot (1-d)^{g-1}\binom{g}{g-1} & (1-d)^{g-1} & & \vdots \\ \vdots & & \ddots & 0 \\ d^g & \cdots & d & 1 \end{bmatrix}$$

Initially all columns in the decoding matrix consist of zero vectors. Therefore the starting pmf $\boldsymbol{s}$ is, $\boldsymbol{s} = [1, 0, ...]$. At least g symbols must be received for decoding to be possible. Hence the estimated number of symbols that must be received in addition to g before all columns contain non-zero values can be evaluated as.

$$\beta \geq \sum_{k=g}^{\infty} \boldsymbol{t}^k(j \neq g), \quad \boldsymbol{t}^k = (\boldsymbol{O}^k \times \boldsymbol{s}) \tag{7}$$

The probability that one column is the zero vector is the probability that one scalar is zero to the power of the number of received symbols. From this we can determine the probability that at least one additional packet is needed when k symbols have been received.

$$\beta(d,g) \geq \sum_{k=g}^{\infty} \left(1 - \left(1 - (1-d)^k\right)^g\right) \tag{8}$$
$$\text{for } 0 < d \leq 1 - q^{-1}$$

D. Linear Dependence Overhead

The total overhead of a given code is given by the expected number of redundant symbols necessary.

$$\frac{\alpha + \beta}{g} \tag{9}$$

To verify Equations (4), (8), and (9) we compare with measured overhead obtained from a high number of runs of our own implementation of RLNC. The results are plotted on Figure 2, where g is on the x-axis and the resulting overhead is on the y-axis.

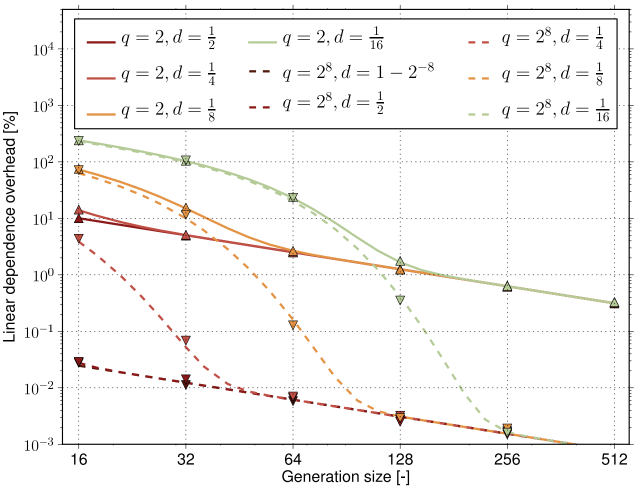


Fig. 2: Linear dependence overhead, analytical values are plotted as lines, measured values are marked with triangles.

On Figure 2, triangles denote measured overhead, which show that the analytical results are a good approximation of the measured values, the error is below 6 % for all measured settings. As g increases the overhead decreases, and when g becomes sufficiently high, d can be decreased with no penalty to the overhead.

III. Coding Vector Representation

To decode a received symbol, a node must in addition to the symbol, hold the corresponding coding vector which results in the *header* overhead. It has been suggested to use a predefined *pseudo random* function to generate coding vectors based on a seed, and then include the seed instead of the coding vector itself, e.g. in [11]. This reduces the overhead to the size of the seed, but also reduces the number of unique coding vectors to the size of the seed. This approach is not suitable for recoding [12]. The reason is that during recoding the coding vector is not drawn randomly but instead computed as $\tilde{\boldsymbol{g}} = \hat{\boldsymbol{g}} \times \boldsymbol{h}$ where $\boldsymbol{h}$ is random. As $\tilde{\boldsymbol{g}}$ can take q^g values, not all possible $\tilde{\boldsymbol{g}}$ can be constructed from the seed. Even if this was possible there is the challenge of identifying which seed produces the wanted coding vector.

We assume that recoding is a requirement, and thus the pseudo random function approach cannot be used, Instead we consider some other representations. A simple but *naive* approach is to construct the coding vector from all the scalars.

s_0	s_1	$\dots$	s_g

Each scalar can be represented by $\log_2(q)$ bits, and there are g such scalars. We denote this overhead introduced by the coding vector γ.

$$\gamma_1 = \log_2(q) \cdot g \tag{10}$$

If the density is low, the coding vector will be sparse, and will mostly consist of 0's. Hence the *naive* approach will be very inefficient. Instead we can represent each non-zero scalar by an index-scalar pair. It is also necessary to append the number of index-scalars pairs, as this can vary.

t	i_0	s_o	i_1	s_1	$\dots$	i_t	s_t

The number of index-scalar pairs, t, takes up at most $\log_2(g)$ bits, as the maximal number of non-zero scalars is g. Each index takes $\log_2(g)$ bits and each scalar takes $\log_2(q)$ bits, and on average there are $g \cdot d$ such pairs. For $q = 2$ it is only necessary to include the indices's as there is only one non-zero scalar.

$$\gamma_2 = \log_2(g) + (\log_2(g) + \log_2(q)) \cdot g \cdot d \tag{11}$$

The coding vector can also be represented by a bit array, that indicates which scalars are non-zero, and the values of these scalars.

$a_0 \quad a_1 \quad \dots \quad a_g$	s_x	s_y	$\dots$	s_z

The bit array can be represented by g bits. Each of the scalars takes $\log_2(q)$ bits, and on average there are $g \cdot d$ such scalars for each encoded symbol. If the bit array is compressed with an optimal code, the amount of bits necessary to represent it can be reduced from g to the entropy of the bit vector, $H(a)$, which can be calculated from d and g.

$$\gamma_3 = H(a) + \log_2(q) \cdot g \cdot d \tag{12}$$

A. Total Overhead

The *total* overhead constitutes the *linear dependence* and *header* overhead, divided by the size of a generation $g \cdot m$

$$\frac{(\alpha + \beta) \cdot m + (g + \alpha + \beta) \cdot \gamma}{g \cdot m} \tag{13}$$

Three examples of the contributions to the total overhead is illustrated on Figure 3. On the x-axis in the range $[10^{-3}, 1]$, on the y-axis is the resulting overhead, and the minimal overhead is marked with a vertical line. On the figure, four contributions from Equation (13) are stacked, $\frac{\alpha}{g}$ from the field size, $\frac{\beta}{g}$ from the density, $\frac{\gamma}{m}$ from the coding vector representation, and the remainder $\frac{(\alpha+\beta)\cdot\gamma}{g \cdot m}$.

On Figure 3 it can be seen that the contribution from α is constant. On Figure 3a the contribution from β is dominating until the density reaches approximately 0.1. When g is larger, in Figure 3b, the contribution from α decreases, and the contribution from β decreases faster. However, the contribution from γ becomes bigger, for high densities. For a higher q, in Figure 3c, the contribution from α is significantly reduced. However, for high densities the contribution from γ dominates.

The interesting result is the minimal obtainable overhead for a given value of g. Therefore we have identified this for different values of g, the result of this search is plotted on Figure 4, where g is on the x-axis and lowest total obtainable overhead is on the y-axis.

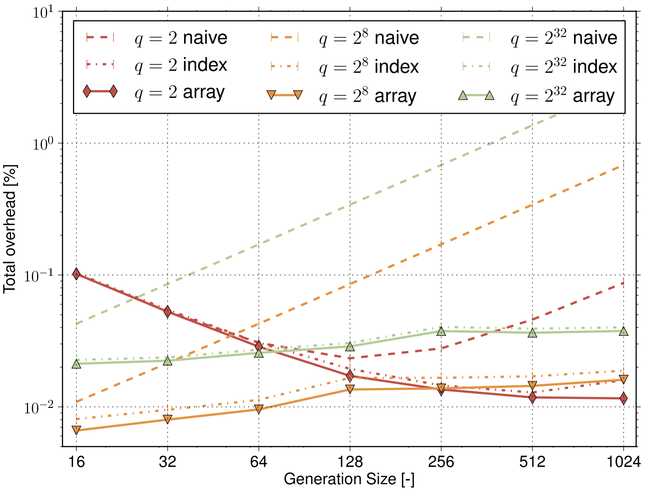
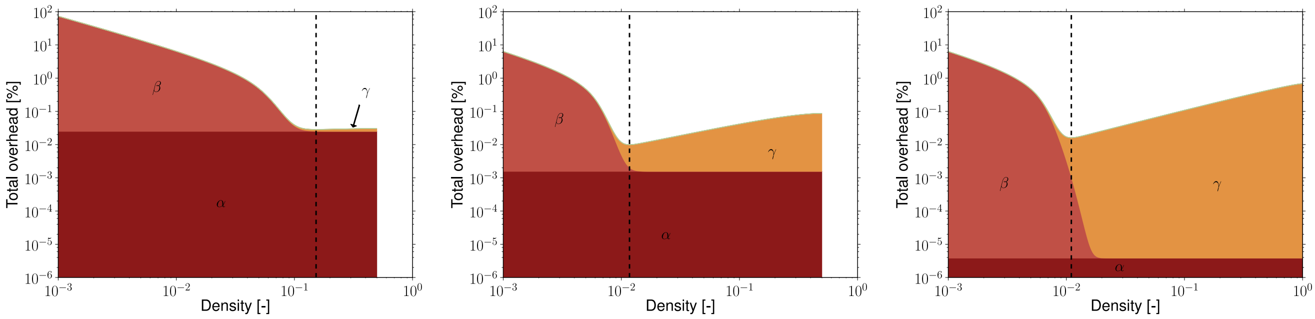


Fig. 4: Lowest total overhead obtainable for different approaches when g is varied.

Interestingly the result shows that $q = 2^{32}$ should never be used. The reason is that the increased entropy of the coding vector is much larger compared to the benefit from the high q-value. For $g < 256$ the lowest overhead can be obtained when $q = 2^8$. For $g > 256$, $q = 2$ can give the lowest overhead.



(a) $q = 2$, $g = 64$

(b) $q = 2$, $g = 1024$

(c) $q = 2^8$, $g = 1024$

Fig. 3: Examples of the total overhead, that is a function of the field size, the density, and the coding vector representation.

One might conclude that a very low generation size would be the best choice. However, it is important to remember the consequences of a low generation size, see Section II-A. Since the *index* approach is much simpler to implement compared to the *array* approach, it may still be useful as the performance when $q = 2$ is similar for the two approaches.

Remark that all evaluations are performed at $m = 1500$ B. This fits well with bulk data distribution or very high rate media streaming over Wireless Local Area Network (WLAN) networks. To evaluate settings where m is significantly different, see [13] for a small script to evaluate the overhead for different setting.

IV. Conclusion

In this paper we have analyzed the transmission overhead of RLNC, as a function of the generation size, field size, density, and coding vector representation. The results have been verified with measurements from our own implementation of RLNC. The results show that generally, in the case where recoding must be supported, a field size of 2 and a low density should be used. If the field size and density is increased, the bits necessary for coding vector representation increases faster than the improvement obtained from the lower amount of linearly dependent packets. However, if the generation size is very low a larger field size than 2 provides the lowest overhead. From a transmission overhead point-of-view, if the recoding operation is not required, the generation size, field size, and density should be chosen as high as possible. However, these parameters also impact the coding throughput, therefore they must be chosen with care in practical applications. As the coding throughput is implementation and topology dependent, no single set of optimal values exists. The results in this work can be used when RLNC is deployed in a real application. The practical performance in terms of coding throughput and energy consumption of the used RLNC implementation, can be compared with the transmission overhead obtained for given parameters. Hence a good trade-off in terms of coding throughput, transmission overhead, and energy can be determined, for a given application and network topology.

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 NOKIA, 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] 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] H. Shojania and B. Li, “Parallelized progressive network coding with hardware acceleration,” in *Quality of Service, 2007 Fifteenth IEEE International Workshop on*, June 2007, pp. 47–55.

[4] X. Chu, K. Zhao, and M. Wang, “Massively parallel network coding on gpus,” in *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, December 2008, pp. 144–151.

[5] 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.

[6] H. Shojania, B. Li, and X. Wang, “Nuclei: Gpu-accelerated many-core network coding,” in *The 28th Conference on Computer Communications (INFOCOM 2009)*, April 2009.

[7] D. Lucani, M. Médard, and M. Stojanovic, “Systematic network coding for time-division duplexing,” jun. 2010, pp. 2403 –2407.

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

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

[10] D. E. Lucani, M. Stojanovic, and M. Médard, “Random linear network coding for time division duplexing: When to stop talking and start listening,” *CoRR*, vol. abs/0809.2350, 2008, informal publication. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr0809.html#abs-0809-2350

[11] C.-C. Chao, C.-C. Chou, and H.-Y. Wei, “Pseudo random network coding design for ieee 802.16m enhanced multicast and broadcast service,” in *Vehicular Technology Conference (VTC 2010-Spring)*, may. 2010, pp. 1 –5.

[12] Z. Liu, C. Wu, B. Li, and S. Zhao, “Uusee: Large-scale operational on-demand streaming with random network coding,” in *IEEE International Conference on Computer Communications*, 2010, pp. 2070–2078.

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

Paper 5

Perpetual Codes for Network Coding

Janus Heide
Morten V. Pedersen
Frank H.P. Fitzek
Muriel Médard

IEEE Transaction on Mobile Computing (2012). submitted.

Perpetual Codes for Network Coding

Janus Heide*, Morten V. Pedersen*, Frank H.P. Fitzek* and Muriel Médard†
* Aalborg University, Aalborg, Denmark, Email: [jah|mvp|ff]@es.aau.dk
† Massachusetts Institute of Technology, Cambridge, Massachusetts, Email: medard@mit.edu

Abstract—Random Linear Network Coding (RLNC) provides a theoretically efficient method for coding. Some of its practical drawbacks are the complexity of decoding and the overhead resulting from the coding vector. For computational limited and battery driven platforms these challenges are particular important as they lead to increased computational load and energy consumption. In this work we present an approach to RLNC where the code is sparse, non-uniform and the coding vectors have a compact representation. 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 presented approach can provide a coding overhead arbitrarily close to that of RLNC, but at significantly reduced computational load. Additionally the approach allows for easy adjustment between coding throughput and code overhead, which makes it suitable for a broad range of platforms and applications.

Index Terms—network coding, implementation, algorithms, complexity

1 Introduction

Network Coding (NC) is a promising paradigm [1] that has been shown to provide benefits in many different networks and applications. NC enables coding at individual nodes in a communication network, and thus is fundamentally different from the end-to-end approach of channel and source coding. With NC packets are no longer treated as atomic entities as they can be combined and re-combined at any node in the network. This allows 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-to-peer or highly mobile networks.

In this work we focus on random NC approaches RLNC [2], and disregard deterministic coding approaches. The reason is that our primary interest is cooperative and highly mobile wireless networks, which fit perfectly with the highly decentralized nature of RLNC. In particular RLNC reduces the signaling overhead and increases robustness towards 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 desirable.

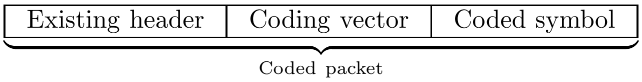
Unfortunately, RLNC is inherently computational demanding which have spawned several efforts to produce optimized implementations and modify the underlying code [3], [4]. Even though several solutions and implementations have been declared to provide *sufficient coding throughput* continued efforts are valid as it can ensure higher coding throughput, which conserve computational resources for other tasks such as video decoding, and reduce the energy consumption introduced by coding. This is of particular importance when NC is deployed on battery driven devices with modest computational capabilities.

This paper presents our work on an alternative approach to RLNC that can provide benefits over standard RLNC. The encoding is sparse and non-uniform which allows for fast decoding as *fill-in* [5] is avoided while at the same time maintaining the possibility of recoding. We describe how encoding and decoding and different types of recoding can be performed. We analyze the overhead and complexity of the proposed approach and verify our results with our own C++ implementation, which also provides practical throughput results. The results show that significantly lower decoding complexity compared to RLNC is possible even at code overheads very close to that of RLNC.The highest gain is obtained at the highest tested generation size of 2048, where both encoding and decoding throughput is approximately one order of magnitude higher than that of standard RLNC. Additionally it is possible to adjust the trade-off between code overhead and coding complexity which makes this approach applicable for a wide range of platforms. Finally it solves the problem of efficient coding vector representation, discussed in [6].

This paper is primarily intended for researchers and developers that work with reliable data distribution on wireless and mobile platforms. Therefore we provide a short overview of RLNC and related work in Section 2. The approach to encoding, decoding and recoding is presented in Section 3 together with algorithms aimed at implementations in C or C++. Section 4 provides some analysis of the performance of the code in terms of decoding complexity and code overhead, and compares measurements results obtained from our implementation with the analytically expressions. Readers primarily interested in theoretical results will already be familiar with with Forward Error Correction (FEC) and in particular RLNC and should therefore skip Section 2 as well as the majority of Section 3, specifically from Section 3.1.

2 RLNC and Related Work

When data is distributed from one or more sources to one or more sinks using RLNC. The data is *encoded* at the sources to produce coded symbols and coding vectors that describe the encoding procedure. Together a coded symbol and a coding vector form a coded packet. When a sink has received *enough* such coded packets, it can *decode* the original data. Additionally at any relaying nodes in between the sources and sinks, received symbols can be recombined and thus *recoded*.



For practical reasons the original data is typically divided into *generations* [7] of size g, we denote the data in such a generation $\boldsymbol{M}$. This ensures that coding can be performed over data of any size and that the performance of RLNC is independent of the data size. Each generation is divided into symbols and these symbols are then combined at random to create a coded symbol. As all operations are performed over a Finite Field (FF) $\mathbb{F}_q$, the code is linear, and thus new valid coded symbols can be created from coded and non-coded symbols. Fig. 1 illustrates how the original symbols can be combined at random and provide a endless stream of coded symbols. The original data can then be decoded by inverting the coding operations performed on the coded symbol. See [8], [9] for an introductions to FF and RLNC.

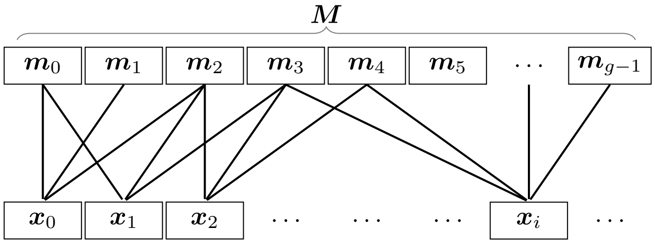


Fig. 1: Coded symbols are created from the original data.

Dividing the data into generations reduces both the computational work and the decoding delay. Unfortunately, it also introduces the need for additional signaling [10], [11], as each of the generations must be decoded successfully before the original data is recovered fully. It also increases the probability that the sink receives linearly dependent symbols which adds to the overhead of the code. For network typologies where it can be assumed that symbols are received only from sources that hold the original information, this overhead is well understood [12]–[14]. In such systems the parameters of the code can be chosen so the overhead tends to zero and can be ignored. However, these parameters present a trade off where generally higher values result in lower code overhead but lower coding throughput. [4]. The coding throughput also depends on less deterministic parameters e.g. the hardware platform, programming language, and implementation optimization's [14]–[18]. Therefore a universally optimal set of values cannot be identified, as it depends on the system in where they are used and on the devices on which it is deployed.

Some simplifications that can increase the coding throughput of RLNC are binary, systematic, and sparse variants [6], [14], [19], [20]. Binary codes are in widespread use and can obtain a low code overhead. They can be fast as operations in the binary field can be performed in parallel by all modern computers.

Using a systematic code comes with no cost in term of overhead, and can potentially provide a high gain in both encoding and decoding throughput. Unfortunately, it is not possible to apply it in general, but only at sources. Thus there is no or little gain if recoding is performed, which is the main reason to use RLNC in the first place. Using a sparse random code provides similar benefits and drawback as a systematic code. It becomes impractical to perform recoding, and the gain in decoding throughput can be small or non-existing [21].

Alternatively the underlying code can be fundamentally modified or replaced to ensure a lower decoding complexity. A noteworthy suggestion is to use a convolutional code as the underlying code [22], [23] as they have been used in communication systems for many years. These efforts are still primarily theoretical as to the best of our knowledge currently no implementation of convolutional codes for NC exists. An approach with a related fundamental concept, combined with a concatenated approach such as that used in the Raptor code [24], was suggest in the unpublished draft [25] and called *perpetual codes*. The authors aim was to propose a cache-friendly rateless erasure code, and therefore did not consider recoding. The fundamental idea in our approach is similar and we have adopted the name *perpetual*, importantly our approach supports recoding and is therefore suitable as a NC code. We note that linear block codes and convolutional codes may in some cases be *equivalent*, as they can describe codes with similar realizations using different terminology [26], [27].

Another direction in the search for improved trade-off between computational work and code overhead was suggested in [28]. Here the authors considered coding over several generations, called a *random annex* code [11], [29]. Each generation is extended to include symbols from other generations and thus when a generation is decoded these extra symbols are released. 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. Importantly the idea of a *random annex* can be applied to many underlying codes, including the perpetual code considered in this work.

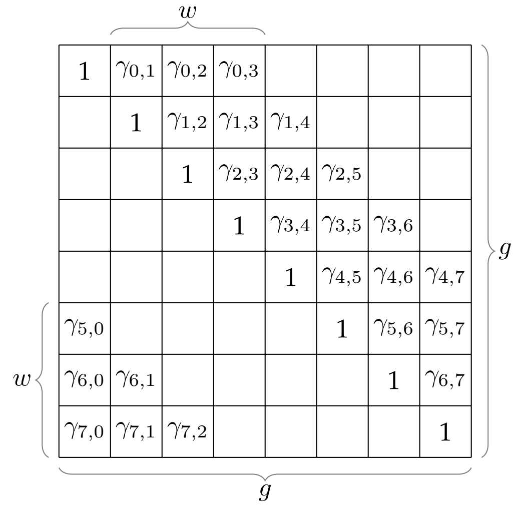
3 Code Operation

This section introduces the code and the three operations, *encoding*, *decoding* and *recoding*, that can be performed at nodes in the network. The notation used for analysis and algorithms is listed Table 1. In vectors and matrices we denote the first element with index zero. In some algorithms the value -1 is used to denote non-valid or non-existing.

TABLE 1: Notation used for analysis and algorithms

Symbol	Definition
g	Generation size
q	Field size
w	Coding vector witdth
$\mathbb{F}_q$	A finite field with q elements
$\boldsymbol{g}$	Coding vector, with g elements starting at element 0
$\boldsymbol{G}$	Matrix containing all received coding vectors
$\boldsymbol{x}$	Coded symbol
$\boldsymbol{X}$	Matrix containing all received symbol
$\boldsymbol{h}$	Local recoding vector
$\boldsymbol{G}_i$	The ith row of the coding matrix
$\boldsymbol{G}_{i,j}$	The index in row i and column j of the coding matrix.
$\boldsymbol{X}_i$	The ith row of the symbol matrix
$\boldsymbol{X}_{i,j}$	The index in row i and column j of the symbol matrix.
$\boldsymbol{g}^p$	The pivot index for the coding vector
$\boldsymbol{g}^b$	The backside element for the coding vector
$\boldsymbol{b}_i$	The row which have backside element i.
ρ and ϱ	Local variables for pivot indices
β	Local variable for backside indices
?	A randomly drawn integer.

In RLNC the elements in the coding vector $\boldsymbol{g}$ are drawn completely at random, and thus each coded symbol is a combination of all the original symbols in one generation. For the perpetual approach we consider in this work, this is not the case. Instead an elements 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. 2 for a small example of some resulting coding vectors.



1	$\gamma_{0,1}$	$\gamma_{0,2}$	$\gamma_{0,3}$				
	1	$\gamma_{1,2}$	$\gamma_{1,3}$	$\gamma_{1,4}$			
		1	$\gamma_{2,3}$	$\gamma_{2,4}$	$\gamma_{2,5}$		
			1	$\gamma_{3,4}$	$\gamma_{3,5}$	$\gamma_{3,6}$	
				1	$\gamma_{4,5}$	$\gamma_{4,6}$	$\gamma_{4,7}$
$\gamma_{5,0}$					1	$\gamma_{5,6}$	$\gamma_{5,7}$
$\gamma_{6,0}$	$\gamma_{6,1}$					1	$\gamma_{6,7}$
$\gamma_{7,0}$	$\gamma_{7,1}$	$\gamma_{7,2}$					1

Fig. 2: All possible coding vectors, when $g = 8$ and $w = 3$. The γ's denote randomly drawn elements from $\mathbb{F}_q$.

3.1 Encoding

The data to be transmitted from the source is divided into generations, we denote the data in such a generation $\boldsymbol{M}$. Each generation is divided into g symbols that are represented with one or more FF elements $\mathbb{F}_q$. The symbols are combined as specified by the coding vector $\boldsymbol{g}$ in order to create coded symbols $\boldsymbol{x}$.

$$\boldsymbol{x} = \boldsymbol{M} \cdot \boldsymbol{g} \tag{1}$$

The construction of a coding vector $\boldsymbol{g}$ and the corresponding coded symbol $\boldsymbol{x}$ is described by Algorithm 1.

Algorithm 1: encode

```
Input: $\boldsymbol{M}$
$\boldsymbol{g} \leftarrow \boldsymbol{0}$
$\rho \leftarrow (? \mod g)$
$\boldsymbol{g}_\rho \leftarrow 1$
for $i \in (\rho, \rho + w]$ do
    $\boldsymbol{g}_{(i \mod g)} \leftarrow (? \mod q)$
$\boldsymbol{x} \leftarrow \boldsymbol{M} \cdot \boldsymbol{g}$
return $\boldsymbol{g}, \boldsymbol{x}$
```

An index in the generation is drawn at random and used as the pivot, $\rho \in [0, g)$. The index in $\boldsymbol{g}$ that corresponds to this pivot element is set to one. For the subsequent w indices in $\boldsymbol{g}$ an element is drawn at random from $\mathbb{F}_q$. The remaining elements in $\boldsymbol{g}$ are zeros. The resulting coding vector is of the form illustrated in Fig. 2. 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.

It is trivial to represent the coding vector in a very compact way. Each coding vector can be represented by an index and w scalars. The necessary bits for their representation is given by Equation (2). The index can take g values and each of each of the w elements can take q values.

p	s_1	s_2	...	s_w

$$|\boldsymbol{g}| = \log_2(g) + w \cdot \log_2(q) \text{ [bits]} \tag{2}$$

Coding vectors can be generated in slightly different ways depending on how ρ is drawn and the size of w, see Table 2. The *systematic* mode does not produce coding vectors of the specified form, but we include it for completeness.

TABLE 2: Different encoding modes.

Mode	ρ drawn	w
Random	random $\in [0, g)$	$0 < w < g$
Sequential	sequentially looping from 0 to g-1	$0 < w < g$
Systematic	sequentially from 0 to g-1, subsequently drawn at random $\in [0, g)$	$w = 0$

3.2 Decoding

A node that receives coded packets can decode the original data, by collecting the coded symbols in $\hat{\boldsymbol{X}}$ and the coding vectors in $\hat{\boldsymbol{G}}$. The original information $\boldsymbol{M}$ can be found as in Equation (3), provided that $\hat{\boldsymbol{G}}$ is invertible and thus have full rank.

$$\boldsymbol{M} = \hat{\boldsymbol{X}} \cdot \hat{\boldsymbol{G}}^{-1} \tag{3}$$

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.

Unlike RLNC and Sparse Random Linear Network Coding (SRLNC), for this perpetual approach the location of the non-zero values are well defined in the coding vector. This makes it possible to decode symbols efficiently and without the problematic fill-in that can be observed during decoding and recoding SRLNC [21].

3.2.1 On-the-fly Decoding

When a new coded packet arrives its coding vector is inserted into the decoding matrix iff. it has a *pivot candidate* that was not previously identified. We distinguish between *pivot* and *pivot candidate* as the element that is used as the pivot is sometimes first found during the final decoding. Otherwise the previously received symbol with the same pivot candidate is subtracted from the new symbol, and the pivot candidate of the new symbol is changed. This is repeated until a new pivot candidate is identified. If the symbols coding vector is reduced to the zero vector the symbol is discarded.

In Fig. 3, three coded packets have been received and their coding vectors inserted into the decoding matrix, the received packets have pivot candidate zero, one, and seven respectively. Subsequently a coded packet with pivot candidate zero is received. This is denoted with a filled circle and arrow pointing to the coding vector of the packet in the left hand side matrix. A symbol with the same pivot candidate have already been identified. Therefore the existing row zero is subtracted from the incoming packet. This is denoted with the arrow pointing left into the left hand side matrix. The element that initially was the pivot candidate is now zero and an element to the right has now become the pivot candidate. This step is repeated for the new pivot candidate and thus row one is subtracted from the incoming packet and element two becomes the pivot candidate. As this pivot candidate was previously not identified the coding vector is inserted into the decoding matrix, which is marked with orange and the arrow pointing right into the decoding matrix.

A special case is when the on-the-fly phase causes the pivot candidate to wrap around to the start of the coding vector. If the last element in the coding vector is reduced to the zero vector, the first element in the vector is considered next and becomes the pivot candidate. An example of this is illustrated in Fig. 4. The incoming packet has pivot candidate seven for which a pivot candidate have already been identified in $\hat{\boldsymbol{G}}$. Thus row seven in $\hat{\boldsymbol{G}}$ is subtracted from the incoming packet. The resulting coding vector has a zero at index seven and thus the pivot candidate is now index zero. The packet is then further reduced similarly to the example in Fig. 3.

Algorithm 2: forwardSubstitute

```
Input: $\boldsymbol{g}, \boldsymbol{x}$
while $\boldsymbol{g} \neq \boldsymbol{0}$ do
    $\rho \leftarrow \boldsymbol{g}^p$
    if $\boldsymbol{G}_\rho \neq \boldsymbol{0}$ then
        $\boldsymbol{g} \leftarrow \boldsymbol{g} \oplus \boldsymbol{G}_\rho$   ▷ Substitute into the new packet
        $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus \boldsymbol{X}_\rho$
    else
        $\boldsymbol{G}_\rho \leftarrow \boldsymbol{g}$   ▷ Insert the new packet
        $\boldsymbol{X}_\rho \leftarrow \boldsymbol{x}$
        return $\rho$
return $-1$
```

In Algorithm 2 unless the received coding vector has been reduced to the zero vector, the existing row with the same pivot candidate is substituted into the received symbol. If a previously not seen pivot candidate is identified the coding vector and symbol is inserted into the respective matrices. Importantly this algorithm guarantees that w is not increased during decoding.

The coding vector can be reduced to the zero vector if it is a linear combination of previously received coding vectors. It is also possible to end in a dead-lock 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. We have determined empirically that decoding should be terminated after $2g$ or $3g$ iterations. To avoid wasting operations on such cases, row operations can first be performed on the coding vector and then repeated on the coded symbol [21]. We note that the resulting overhead in both cases are due to the same reason, namely that the symbol is a linear combination of already received symbols.

A simple optimization in cases where the w of the incoming packet is lower than the w of the held symbol with the same pivot candidate, is to simply swap these two to guarantee that w is never increased. Our current implementation does not support this and we leave it to future work to test whether this increases the decoding throughput. However, some of our previous experiments showed that such optimizations often comes with a high cost in terms of bookkeeping [21].

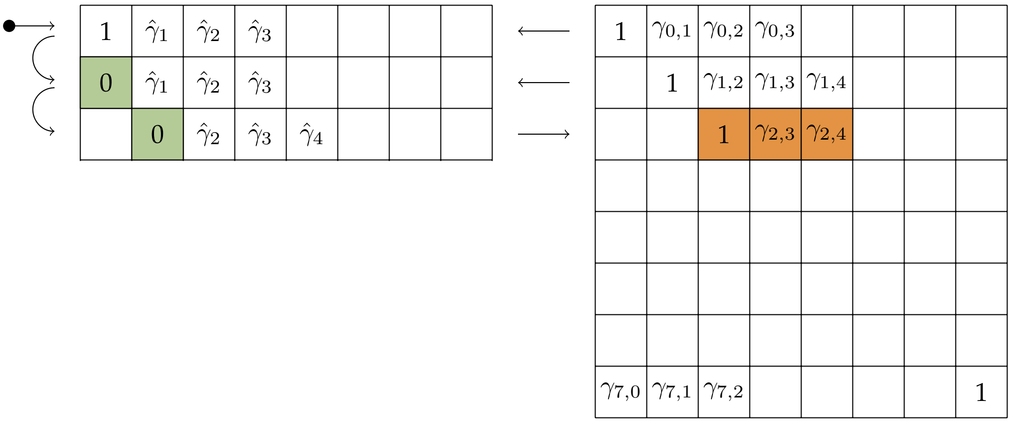


Fig. 3: *On-the-fly* decoding of a received coded packet. The right hand side matrix is the decoding matrix $\hat{\boldsymbol{G}}$. The left hand side matrix show the coding vector of the incoming symbol as it is decoded. The γ's denotes random field elements. The filled circle and arrow indicate the original coding vector of the incoming packet. The straight arrows indicate what rows are substituted into the coding vector. The arching arrows indicate the step of the decoding.

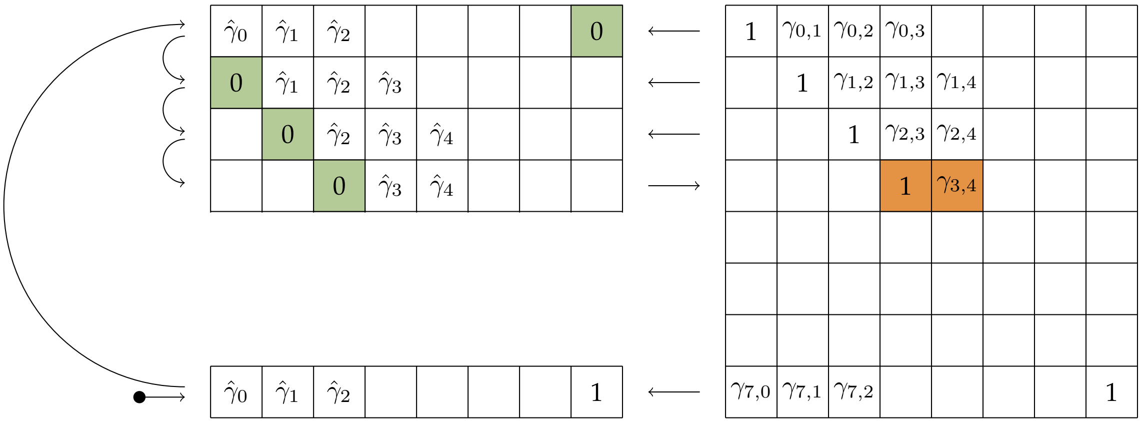


Fig. 4: *On-the-fly* decoding similar to Fig. 3, but the pivot candidate wraps around the end of the decoding matrix.

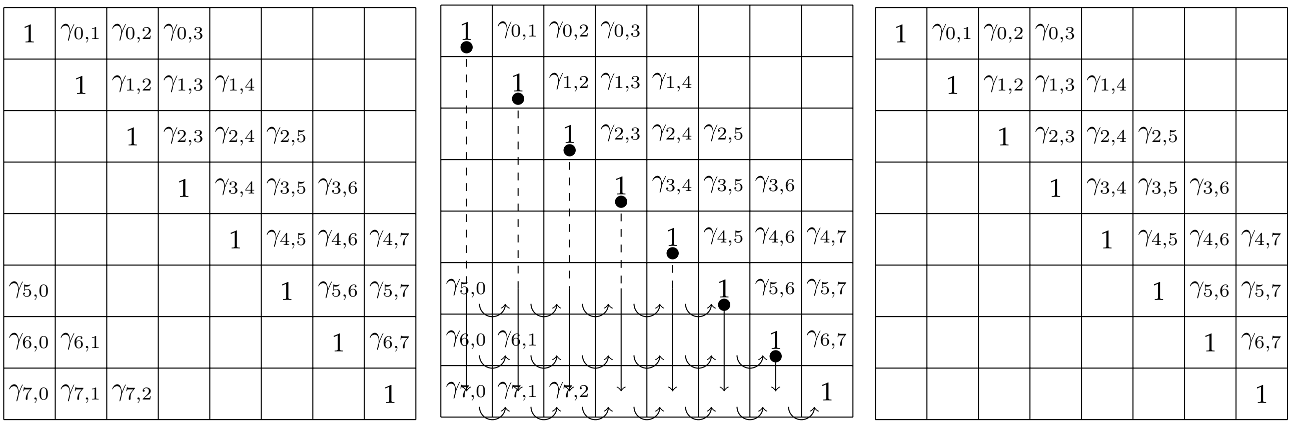
3.2.2 Final Decoding

When a pivot candidate has been identified for all rows, final decoding is performed by forward substitution and backwards substitution. Initially the decoding matrix has a form similar to that shown in Fig. 5a. 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 also be noted that even though a pivot candidate has been identified for all rows this does not guarantee that the decoding matrix has full rank. Therefore it is important to perform the final decoding in a way that ensures that the decoding matrix is not left in a state where future decoding becomes impossible or problematic.

To bring the matrix onto echelon form forward substitution is performed on the non-zero elements in the lower left corner of Fig. 5a. When forward substitution is performed on the first column, non-zero elements can be introduced in the lower w rows and further substitution becomes necessary, illustrated on Fig. 5b. After the forward substitution step the decoding matrix is brought onto echelon form in Fig. 5c.

The final forward step can be implemented in several ways. The approach in Algorithm 3 ensure that the decoding matrix is always left in a valid state also if partial final decoding occur. This happens in cases where it turns out that the decoding matrix does not have full rank, even though a pivot candidate for each row was identified.

In Algorithm 3 for each column (i) a pivot element must be defined, if no such pivot element can be found it means that none of the received symbols can be used to decode the corresponding row, and we need to receive additional symbols. Therefore we traverse all the rows (j) from the diagonal and down, as we know that for all rows above a pivot index have already been identified. When we find a row for which the pivot we are looking for is non-zero we swap it to the correct row, if it not already located correctly. We then forward substitute into the below rows. If we iterate to the last row without identifying a pivot element we cannot decode the row we are looking for, and we discard the symbol that incorrectly is located on row i. However, we do not want to discard useful symbols, therefore we check if the



(a) Initial

(b) During final forward substitution

(c) After final forward substitution

Fig. 5: The decoding matrix $\hat{\boldsymbol{G}}$ at various states of the final decoding. The dotted part of the arrows indicate rows where no substitution is needed. The arching arrows show how the pivot candidate moves towards the diagonal.

Algorithm 3: finalForward

```
for $i \in [0, \ldots, g)$ do
    for $j \in [i, \ldots, g)$ do
        if $\boldsymbol{G}_{j,i} \neq 0$ then
            if $i \neq j$ then
                $\boldsymbol{G}_i \leftrightarrow \boldsymbol{G}_j$
                $\boldsymbol{X}_i \leftrightarrow \boldsymbol{X}_j$
            for $k \in [\max(j+1, g-w), g)$ do
                if $\boldsymbol{G}_{k,i} \neq 0$ then
                    $\boldsymbol{G}_k \leftarrow \boldsymbol{G}_k \oplus \boldsymbol{G}_i$
                    $\boldsymbol{X}_k \leftarrow \boldsymbol{X}_k \oplus \boldsymbol{X}_i$
            break    ▷ Found a pivot, skip to the next
        else
            if $j = (g-1)$ then
                if $i \neq (g-1)$ then
                    if $\boldsymbol{G}_i \neq \boldsymbol{G}_{i+1}$ then
                        $\boldsymbol{G}_{i+1} \leftarrow \boldsymbol{G}_{i+1} \oplus \boldsymbol{G}_i$
                        $\boldsymbol{X}_{i+1} \leftarrow \boldsymbol{X}_{i+1} \oplus \boldsymbol{X}_i$
                $\boldsymbol{G}_i \leftarrow \boldsymbol{0}$
                $\boldsymbol{X}_i \leftarrow \boldsymbol{0}$
```

coding vector for row i is equal to the row below, if not we simply add it to the row below. If the two symbols were identical the result would be the **0** vector and we would discard two rows instead of one. Finally if we are looking for the pivot element for the final column, there are no rows below our target row, and we simply discard without checking. In this way the decoding matrix will always be brought as close to echelon form as possible.

After the forward substitution, and when the rank of the matrix is full, standard backward substitution is performed to bring the decoding matrix to reduced echelon form and decode the original data.

Algorithm 4: finalBackward

```
for $i \in (g, \ldots, 0]$ do
    for $j \in (i, \ldots, \max(i-w, 0)]$ do
        if $\boldsymbol{G}_{j,i} \neq 0$ then
            $\boldsymbol{G}_j \leftarrow \boldsymbol{G}_j \oplus \boldsymbol{G}_i$
            $\boldsymbol{X}_j \leftarrow \boldsymbol{X}_j \oplus \boldsymbol{X}_i$
```

All rows from the bottom and up is used to remove any remaining non-zeros in the rows above. Note that for each index it is only necessary to inspect the above w rows, as all other rows are guaranteed to be zero due to the form of the decoding matrix. Algorithms 2-4 can be combined to create the decoder in Algorithm 5.

Algorithm 5: decode

```
Input: $\boldsymbol{g}, \boldsymbol{x}$
$\rho \leftarrow$ forwardSubstitute($\boldsymbol{g}, \boldsymbol{x}$)
if $\rho \neq -1$ then
    $\boldsymbol{G}_\rho \leftarrow \boldsymbol{g}$
    $\boldsymbol{X}_\rho \leftarrow \boldsymbol{x}$
if rank($\boldsymbol{G}$) $= g$ then
    finalForward()
if $\boldsymbol{G}_{\text{rank}} = g$ then
    finalBackward()
return rank($\boldsymbol{G}$)
```

When a new packet arrives it is first forward substituted. If a new pivot element is identified the coding vector and coded symbol is inserted into the decoding matrix. When the rank of the decoding matrix is full final decoding is attempted using forward substitution. This might initially fail, however when it succeeds final backwards substitution is performed and the original data in the generation is decoded.

3.2.3 Partial backwards substitution

To decrease the final decoding work and thus the final decoding delay we can modify the decoding process. The overall goal is to perform some backward substitutions, but without causing fill-in. We define the *backside* as the last non-zero element after the pivot candidate, note that if $p \geq g - w$ then the backside index can be lower than the pivot index. Here we consider an approach where the backside is moved towards the pivot candidate and the non-zero part of the vector is decreased.

Consider Fig. 3, we see that after the new coding vector have been inserted, two rows have the same backside, row one and two. Thus if we substitute row two from row one the current backside element of row one will become a zero. Note that it may be necessary to multiply row two with a scalar to insure that the backside of the two rows have the same value. As the pivot element of row two is smaller than that of row one it is guaranteed that the pivot element of row one will not be altered. By using this insight we can perform a partial backward substitution as illustrated in Fig. 6. When the first partial backward substitution have been performed, the backside of row one is decreased, and now be the same as that of row one. Hence we can continue by subtracting row one from row zero, and so on.

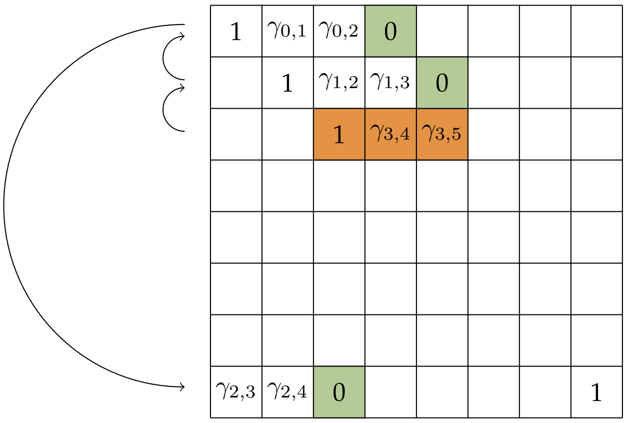


Fig. 6: Partial backward substitution after partial forward substitution of a received packet. The zeros indicate the reduced elements.

In Fig. 6 row two is substituted into row one and thus the former backside element of row one becomes zero. Then row one is substituted into row zero to reduce the backside of row zero. Now the top of the decoding matrix have been reached and therefore row zero is substituted into row seven. The width of the non-zero diagonal in the decoding vector has been decreased as indicated by the introduced zero's.

Algorihtm 6 provide a way to implements this, based on maintaining a list of the row that have a given backside. If two rows have the same backside, one should be subtract from the other and they would no longer have the same backside.

Algorithm 6: reduceBack

```
   Input: $\boldsymbol{g}, \boldsymbol{x}, \rho$
1  while 1 do
2      $\beta \leftarrow \boldsymbol{g}^b$                                ▷ The target backside
3      $\varrho \leftarrow \boldsymbol{b}_\beta$              ▷ Existing row with target backside
4      if $\varrho = \rho$ then
5          return                                              ▷ If itself stop
6      if $\varrho = -1$ then
7          $\boldsymbol{b}_\beta \leftarrow \rho$        ▷ A unique backside was identified
8          return
9      else
10         if $(\beta - \rho \mod g) < (\beta - \varrho \mod g)$ then
11             $\boldsymbol{G}_\varrho \leftarrow \boldsymbol{G}_\varrho \oplus \boldsymbol{g}$
12             $\boldsymbol{X}_\varrho \leftarrow \boldsymbol{X}_\varrho \oplus \boldsymbol{x}$
13             $\boldsymbol{b}_\beta \leftarrow \rho$
14             $\rho \leftarrow \varrho$                 ▷ New target row identified
15             $\boldsymbol{g} \leftarrow \boldsymbol{G}_\varrho$
16             $\boldsymbol{x} \leftarrow \boldsymbol{X}_\varrho$
17         else
18             $\boldsymbol{g} \leftarrow \boldsymbol{g} \oplus \boldsymbol{G}_\varrho$
19             $\boldsymbol{x} \leftarrow \boldsymbol{x} \oplus \boldsymbol{X}_\varrho$
```

Whenever a row is inserted into the decoding matrix we check if there existing a row with the same backside. If not we are done otherwise we substitute the row with the lowest pivot into the row with the highest pivot. The row we substitute into will now have a new backside and we continue the process with this row. In line ten we test to see if the new row is shorter than the row that is currently in the backside list. If this is the case we subtract the new row from the old row, insert the new row into the backside list, and continue to reduce the old row, otherwise we subtract the old row from the new row.

Finally we can define an alternative decoder that utilizes the partial backward substitution approach.

Algorithm 7: decode *with partial backwards substitution*

```
Input: $\boldsymbol{g}, \boldsymbol{x}$
$\rho \leftarrow$ forwardSubstitute($\boldsymbol{g}, \boldsymbol{x}$)
if $\rho \neq -1$ then
    reduceBack($\boldsymbol{g}, \boldsymbol{x}, \rho$)
    $\boldsymbol{G}_\rho \leftarrow \boldsymbol{g}$
    $\boldsymbol{X}_\rho \leftarrow \boldsymbol{x}$
if $\mathrm{rank}(\boldsymbol{G}) = g$ then
    finalForward()
if $\mathrm{rank}(\boldsymbol{G}) = g$ then
    finalBackward()
return $\mathrm{rank}(\boldsymbol{G})$
```

Another partial backward substitution approach would be to backwards substitute with a row once it becomes fully decoded.

3.3 Recoding

When two or more coded or non-coded symbols have been received they can be combined by recoding. This can be described by Equation (4) and (5), where the collected coding vectors and coded symbols are combined as defined by $\boldsymbol{h}$ of length g', where g' is the number of received symbols. Then $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{g}}$ together form a recoded packet.

$$\tilde{\boldsymbol{g}} = \hat{\boldsymbol{G}} \cdot \boldsymbol{h} \tag{4}$$

$$\tilde{\boldsymbol{x}} = \hat{\boldsymbol{X}} \cdot \boldsymbol{h} \tag{5}$$

In classical RLNC coded packets are accumulated and recoding is performed as a separate operation which result in a significant computational load, we denote this type of recoding *active* recoding. As explained in [6] this form of recoding is not suitable when the code is sparse, because the recoded symbol will be more dense with high probability [5], [21]. To combat this problems we introduce a new type of recoding and denote it *passive* recoding.

3.3.1 Active Recoding

Combining all collected packets completely at random, as in standard RLNC, result in recoded packets where the non-zero elements are no longer confined to w elements. If we instead we can pick packets that have similar pivot elements 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. Unfortunately such an approach significantly increases the complexity of recoding as it introduces a search for an appropriate sets. Additionally it is more deterministic than the standard recoding approach and thus great care must be taken to avoid generating more linearly dependent symbols.

3.3.2 Passive Recoding

When *on-the-fly* decoding is performed, previously received symbols are subtracted from an incoming symbol, to partially decode it. This combining of packets can also be considered as recoding and therefore the operations can be reused in order to reduce the computational load of recoding.

If the operations performed on the received symbols are tracked, a symbol where a sufficient number of operations have been performed can be used as a recoded symbol. 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. It is important to remember that 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.

3.3.3 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.

3.3.4 Re-encoding

When a receiver has decoded a generation it can encode packets the same way as the original source. This is sometimes referred to as recoding, which we believe is misleading, and instead denote this re-encoding to distinguish this from encoding at the original source.

3.3.5 Implementation

In our implementation we have chosen to implement a simple version of the *active plus passive recoding*. The primary reasons is to reuse the operations performed during decoding, and at the same time allow recoding to be performed when and as much as desired. Another important consideration is to avoid introducing a deterministic behavior when recoding.

Algorithm 8: recode

```
Input: $\hat{G}, \hat{X}$
if $\mathrm{rank}(\hat{G}) = 0$ then
    return $-1$                                  ▷ no data available
$\rho \leftarrow (? \mod g)$
while $\hat{G}_{\rho,\rho} = 0$ do
    $\rho \leftarrow (? \mod g)$
$h_p \leftarrow 1$
for $i \in (\rho, \rho + w]$ do
    if $\hat{G}_{i,i} \neq 0$ then
        $h_{(i \mod g)} \leftarrow (? \mod q)$
$\tilde{g} \leftarrow \hat{G} \cdot h$
$\tilde{x} \leftarrow \hat{X} \cdot h$
return $g, x$
```

First row indices are drawn at random until a row that is non-zero is identified. Then for each of the following w rows that are non-zero, a random index is drawn which defines the recoding vector $\boldsymbol{h}$. The remaining indices in $\boldsymbol{h}$ are zeros. The new coding vector and coded symbol are then computed as $\tilde{\boldsymbol{g}} = \hat{\boldsymbol{G}} \cdot \boldsymbol{h}$ and $\tilde{\boldsymbol{x}} = \hat{\boldsymbol{X}} \cdot \boldsymbol{h}$, respectively.

This approach ensure that the width of the recoded vector $w' < 2w$. However, it is worth observing that if a symbol with a higher w is received, the size of w can be reduced during the forward substitution. This would happen more frequently if the width of the received and existing row is compared as mentioned in Section 3.2.

4 Analysis and Experiments

In this section we present analytic and experimental results on the code overhead, complexity and throughput. To verify the analytically expressions we have implemented the proposed code in C++ [30]. This also provide us with the possibility to report on encoding and decoding throughput with is a more interesting parameter as it defines the computational load at the coding nodes. The current implementation is well tested and we believe it provide a good trade-off between simplicity and throughput. As the code is available under a research friendly license, we encourage suggestions that can improve the throughput or simplify the implementation.

4.1 Overhead

The code overhead depends on the field size, density, generation size and possibly other factors. From standard RLNC we have a lower bound for the code overhead as defined in Equation (7), see [6]. The same lower bound holds here, as the lowest overhead is obtained when $w = g - 1$, in which case the perpetual code becomes identical to RLNC.

Equation (6) 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 linearly 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.

$$\alpha \geq \sum_{g'=0}^{g-1}\left(\left(1-\frac{1}{q^{g-g'}}\right)^{-1}-1\right) \tag{6}$$

$$= \sum_{g'=0}^{g-1}\left(\frac{1}{q^{g-g'}-1}\right) \tag{7}$$

Each coded perpetual symbol is a combination of $w+1$ symbols, and there are g pivots they can cover. Each pivot is not already covered with probability $1-\frac{r}{g}$, where r is the rank. The expected probability that a symbol covers a not already seen pivot is calculated by summing over all possible ranks, and divide by the number of possible ranks. From this the probability that a symbol is covered when x coded symbols have been received can be found by the probability that all x coded symbols do not cover the symbol. In the worst case decoding is possible when all g pivots are covered.

$$F_X(x) \geq \left(1-\left(1-\frac{1}{g}\sum_{r=0}^{g-1}(1+w)\cdot g\cdot\left(1-\frac{r}{g}\right)\right)^x\right)^g \tag{8}$$

$$= \left(1-\left(1-\frac{1+w}{g\sum_{r'=1}^{g}\frac{1}{r'}}\right)^x\right)^g \tag{9}$$

The resulting *cdf* can then be used to calculate an upper bound for the code overhead by evaluating the corresponding *survival function (sf)*, which defines the probability that all symbols are not covered after x transmission and thus additional transmissions are necessary.

$$\beta \leq \sum_{x=g}^{\infty} S_X(x) = \sum_{x=g}^{\infty} 1 - F_X(x) \tag{10}$$

$$\alpha \leq O \leq \alpha + \beta \tag{11}$$

In Fig. 7 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 upper and lower bounds respectively.

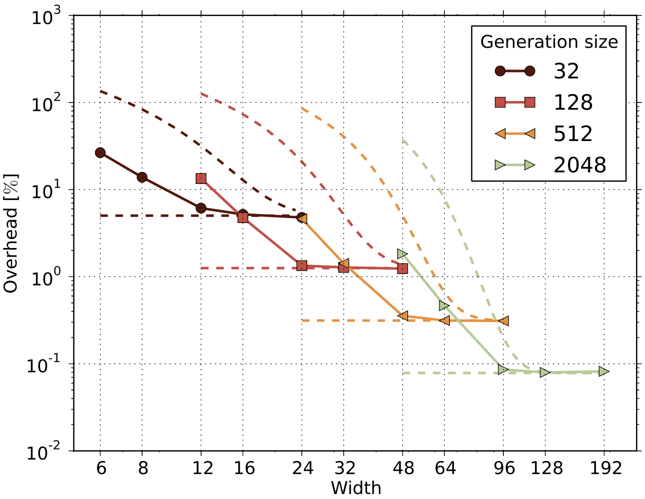


Fig. 7: Code overhead in percent as a function of g and w. The dotted lines denote the analytically lower and upper bounds respectively.

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 what is sufficient 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 becomes tighter as w increases. Thus the provided bounds are useful for identifying a value of w that is sufficiently high.

We note that these result does not follow the overhead as a function of the density defined in [6], which is similar to the width for this code. This is not surprising as the code treated here is significantly less random compared to the sparse RLNC considered in the reference.

4.2 Complexity

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.

To *encode* a single packet, the expected number of row operations is given by Equation (12). 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 $1 - \frac{1}{q}$.

$$1 + w \cdot (1 - \frac{1}{q}) \tag{12}$$

During *on-the-fly* decoding, forward substitution is performed on the incoming symbol until a new pivot candidate is identified, or the symbol is reduced to the $\mathbf{0}$ vector. The forward substitution continues until the next element in the coding vector is non-zero, with probability $1 - \frac{1}{q}$ and has not already been identified as a pivot, with probability $1 - \frac{r}{g}$ where r is the current rank of the decoding matrix. Hence the expected number of indices that must be inspected and hence the number of row operations necessary is found by summing the inverse of this probability and divide with g to find the expected number of operations per packet.

$$\delta_{\text{fly}} \leq \frac{1}{g} \sum_{r=0}^{g-1} \left(\left(1 - \frac{1}{q}\right) \left(1 - \frac{r}{g}\right) \right)^{-1} \tag{13}$$

$$= \frac{q}{q-1} \cdot \frac{1}{g} \sum_{r=0}^{g-1} \frac{g}{g-r} \tag{14}$$

$$= \frac{q}{q-1} \sum_{r'=1}^{g} \frac{1}{r'} \tag{15}$$

For the upper bound for the *final decoding* we consider the worst case, where most scalars are non-zero, see Fig. 5a. The final forward stage on Fig. 5b can be considered in two steps. First the bottom w rows are reduced, so only the last w elements are non-zero, by substituting the top $g-w$ into them, hence Equation (16). Then the bottom w rows are brought onto echelon form. The $(g-w)$'th row 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, hence Equation (18). To include the probability that an element in $\mathbb{F}_q$ is equal to zero, we multiply with $\left(1 - \frac{1}{q}\right)$ and divide by g to find the operations per packet, which is rewritten as $\left(\frac{q-1}{q \cdot g}\right)$.

$$\delta_{\text{forward1}} \leq \left(\frac{q-1}{q \cdot g}\right) \cdot (g-w) \cdot w \tag{16}$$

$$\delta_{\text{forward2}} \leq \left(\frac{q-1}{q \cdot g}\right) \cdot \sum_{i=1}^{w-1} i \tag{17}$$

$$= \left(\frac{q-1}{q \cdot g}\right) \cdot \frac{w \cdot (w-1)}{2} \tag{18}$$

To finalize the decoding a similar procedure is performed, but this time upwards. Each of the $g-w$ bottom rows are substituted into the w rows directly above them. Thus the number of operations is exactly the same as in Equation (16) and Equation (18) and we obtain Equation (20).

$$\delta \leq \delta_{\text{fly}} + 2\delta_{\text{forward1}} + 2\delta_{\text{forward2}} \tag{19}$$

$$= \frac{q}{q-1} \sum_{r'=1}^{g} \frac{1}{r'} + \left(\frac{q-1}{q \cdot g}\right) (w \cdot (2g - w - 1)) \tag{20}$$

In Fig. 8 the calculated and measured 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 on the y-axis. The operations during the two phases are stacked to show the total number of row operations.

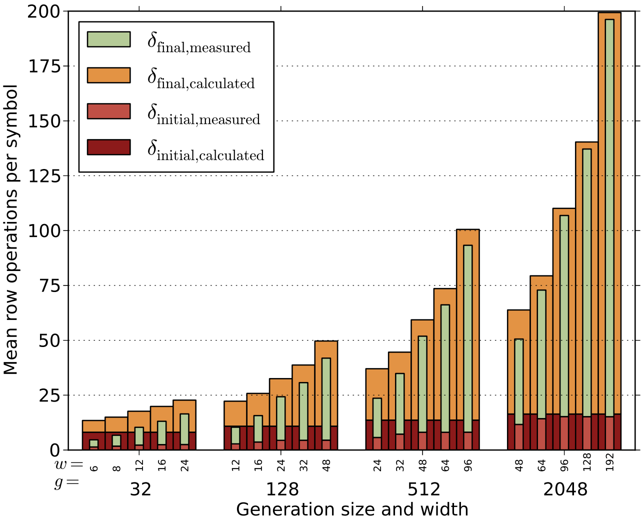


Fig. 8: Mean row operations per decoded symbol.

Both the analytically expression for the *on-the-fly* and *final* provide good bounds for the measured results, 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.

These values can be compared with traditional RLNC where the expected operations to decode a packet is approximately $g/2$ for the binary case [21]. Thus the reduction in complexity compared to RLNC grows as g grows.

4.3 Throughput

A low complexity does not guarantee a low computational load and therefore we investigate the coding throughput. Some reasons are the complexity introduced by the algorithms that determine how decoding is performed, the quality of their implementation and the bookkeeping they add. Other reasons are architecture specific such as cache misses, memory delay and throughput, which additionally can be influenced by the memory access pattern.

For each setting first data was encode, and subsequently decoded on the same machine. Each setting was run at a minimum of 60 s to reduce the deviation.

TABLE 3: Specifications of the test machine.

Model	Dell Opiplex 790 DT
CPU	Intel Core i7-2600 @ 3.40GHz, 8192 KB L2 cache
Memory	16GB DDR3 1333 MHz Dual channel
Chipset	Intel Q65 Express
OS	64bit Debian Wheezy
Compiler	GNU G++ 4.6

To provide a comparison we have performed benchmarks of our RLNC implementation using the same values of g as for the perpetual code. The encoding and decoding throughput's are listed in Table 4 along with the corresponding gains over comparable RLNC in percent.

TABLE 4: Measured encoding and decoding throughput for RLNC and the proposed perpetual code. Throughputs are reported for different generation sizes, g, and in the case of the perpetual code at different widths, w. The lowest tested value of w where the perpetual code provides a similar code overhead as RLNC is marked.

g	w	Overhead [packets]	Encoding [MB/s]	Gain [%]	Decoding [MB/s]	Gain [%]
32	6	8.34	2883.91	156	2879.99	147
	8	4.20	2512.14	123	2034.45	74
	12	1.85	1915.58	70	1359.66	16
	16	1.52	1506.90	34	1214.37	4
	24	1.62	1080.19	-4	881.03	-25
	RLNC	1.61	1126.16	-	1167.67	-
128	12	17.07	1612.36	441	1209.09	326
	16	6.06	1309.09	339	951.04	235
	24	1.77	951.04	219	620.30	119
	32	1.61	742.68	149	526.43	86
	48	1.64	520.06	74	360.34	27
	RLNC	1.61	298.28	-	283.69	-
512	24	24.54	747.29	921	490.50	655
	32	7.03	612.69	737	392.39	504
	48	1.64	449.39	514	273.00	320
	64	1.65	354.48	384	228.40	251
	96	1.56	249.25	241	167.27	157
	RLNC	1.61	73.17	-	64.99	-
2048	48	36.43	314.79	1656	203.11	1321
	64	9.02	263.36	1369	167.82	1074
	96	1.78	198.93	1010	129.93	809
	128	1.66	160.38	795	99.90	599
	192	1.72	115.00	541	66.81	368
	RLNC	1.61	17.93	-	14.29	-

As expected the throughput for both encoding and decoding decreases as g and w increases. The gain increases for higher values of g which corresponds with the analytically results. To ease the comparison with standard RLNC the throughputs for the proposed approach where the code overhead is similar to RLNC is marked. The highest gain in encoding and decoding throughput is observed at the highest tested generation size of 2048, and approximately eleven and nine times that of RLNC respectively.

To make a fair comparison with RLNC we must consider both the overhead and the complexity / throughput simultaneously, as the performance of the perpetual code is a trade-off between overhead and speed. As the lower bounds are the same as for RLNC we can never hope to achieve a lower code overhead. However, we can achieve a similar overhead but at lower computational complexity.

The values of marked values in Table 4 corresponds to the lowest tested w where the code overhead of RLNC and the perpetual code are similar, see Fig. 7. The increase in throughput over the RLNC is significant in particular for higher values of g. It should also be noted that using an excessive high w should be avoided as it decrease the throughput without reducing the code overhead. Additionally a higher w increases the size of the coding vector representation, see Equation (2), which adds to the overall overhead.

For low values of g the increase in throughput is small, and in a single case negative. This is a consequence of the slightly more complicated algorithms compared to the ones used in the standard RLNC implementation. Additionally as g decreases w approaches g if we wish to maintain a similar code overhead as RLNC. Thus the perpetual approach is most useful for values of g that are not very small, as in such cases it can deliver a similar code overhead as RLNC at significantly decreased coding complexity.

Ideally all decoding should be performed *on-the-fly* as this decreases the final decoding delay and distribute the processing load evenly. At the same time decoding should be performed in such a way that *fill-in* does not occur, as this reduces the amount of work necessary to decode [5]. In our presented results the ratio of operations performed in the *on-the-fly* phase is low, see Fig. 8. Fortunately the structure of the code make it possible to perform something 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 proved to be more difficult to analyze and due to space constraint we have omitted it.

We note that the implementation does not take advantage of multiple cores. This could however be exploited by encoding multiple streams simultaneously or encoding simultaneously from different blocks of the same data.

5 Discussion

In this section we discuss some of the choices made in this work, and provide some insights and observations. By collecting these here we hope to provide a story that is more easy to follow in the rest of this paper.

When the performance of the presented and other RLNC codes are investigated, a fundamental parameter is the generation size g. It is important to remember that the generation size defines the lowest delay that can be achieved from when the first packet in a generation is received until the generation is decoded. Therefore the generation size should be defined before any other parameter based on application under consideration. For delay sensitive applications like audio and video streaming this means that low to medium sizes of g, e.g. 32-128, should be used, depending on the bitrate and the size of the payload in each packet. For non-delay type applications such as standard file downloads and p2p g should be as high as can be supported to reduce the code overhead and signaling. However, for high values of g, e.g. 512-2048, the throughput of existing RLNC approaches is low.

In this work we have focused a solution that uses the binary field, $q = 2$. The reason is that the binary field allows for simpler and faster implementations compared to when higher field sizes are used. Additionally higher field sizes are most applicable at low generation sizes as the overhead due to the coding vectors can become much higher than the overhead due to linear dependency.

We have throughout this work compared the proposed perpetual code with RLNC and SRLNC and not other rateless FEC codes. The reason is that NC codes allow for recoding where traditional FEC codes do not, thus they are less suitable for realizing the cooperative networks that is our interest. Compared to RLNC and SRLNC the perpetual code provides the following benefits, depending on the chosen values of g and w.

1) Faster encoding, recoding, and decoding.
2) Sparsity is retained when recoding.
3) Small coding vector representation.
4) Maintain simple decoding algorithms.

As the code is sparse fast encoding is trivial. Fast decoding is possible due to the structure of the code, that make it possible to avoid *fill-in* during decoding. Recoding can be performed fast by using the suggested *passive* recoding approach, we note that this trick can also be employed for other NC codes.

The structure and density of the coded packets can be retained if recoding is performed more carefully than what has been proposed for the standard RLNC, we have denoted this *active* recoding. We note that doing so limits the degrees of freedom when recoding, but believe that our proposed *passive plus active* recoding presents a good trade off between these two approaches.

As the location of the non-zeros are well defined, it is trivial to create compact representations of the coding vectors. We believe that this is important as the commonly used assumption of a pseudo random function can be used to compress the coding vector cannot be used if recoding is to be supported [6]. Thus the size of the coding vector must be included in the total overhead.

The presented decoding algorithms are slightly more complicated than for standard RLNC. However due to the sparsity and structure of the coded symbols it is possible to eliminate many of the inspections that are necessary when inverting the coding matrix.

Our current implementation is an extension to our existing implementation work on RLNC. Therefore we have chosen to use a matrix representation of the decoding matrix and keep the local representation of the coding vectors in their full vector form. This is a simple approach that allow for very fast additions of coding vectors using paralleled instructions. The drawbacks are that coding vectors must be convert between their compact and full representation when coded packets are sent and received. Additionally a large number of zero elements are added which could become a dominating factor for the throughput for very high values of g. Alternatively the coding vectors could be in their compact form locally. However, in this case it would become necessary to perform bit shifts in order to align vectors, so that the additions of vectors can be performed using paralleled instructions.

We have only consider the *random* encoding mode, meaning that the pivot element is always drawn at random and independently of the previous pivots, see Table 2. 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 sequential mode could be used which would decrease the code overhead and in particular the decoding complexity.

The approach presented here is similar to what is called a *smooth perpetual code* in [25], but with two significant differences, we do not use zero padding nor a precode. Because we do not zero pad, 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 generally makes sense to first consider the simplest 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 in fountain codes.

6 Conclusion

In this paper we presented our initial findings on *perpetual codes* which is suitable for RLNC. We described how encoding, decoding, and recoding can be performed and formalized it as implementation near algorithms. We provided initial analysis of the code performance in terms of overhead and complexity. The analysis was compared with measurements obtained from our C++ implementation, from which we also obtained coding throughput measurements.

The analysis and tests showed that the proposed approach can obtain a coding overhead similar to RLNC but at a much lower computational cost. For all tested settings resulting in a code overhead similar to that of RLNC the proposed approach led to improved encoding and decoding throughput. For the highest tested generation size of 2048 the decoding throughput was almost one order of magnitude higher than that of RLNC. Additionally the approach provides an easily adjustable parameter that allow the trade-off between coding complexity and code overhead to be tweaked to the used platform and application.

For the future more rigorous analysis of the code overhead is necessary, especially for the case where low values of w is used. Such analysis would be useful when more advanced variants of the perpetual code is studied. For our implementation we plan to perform tests using higher field sizes, implement and evaluate the partial backward substitution, and perform benchmarks on mobile platforms such as mobile phones. This could help to improve the understanding of how to choose good parameters, and demonstrate the proposed solutions validity on devices in one of the fastest growing categories, mobile phones and tablets.

Acknowledgment

This work was partially financed by the CONE project (Grant No. 09-066549/FTP) granted by 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.

[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.

[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] S. Ingram. (2006) Minimum degree reordering algorithms: A tutorial. Retrieved March 2010. [Online]. Available: www.cs.ubc.ca/~sfingram/cs517_final.pdf

[6] 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.

[7] 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.

[8] A. Neubauer, J. Freudenberger, and V. Kuhn, *Coding Theory: Algorithms, Architectures and Applications*. Wiley-Interscience, 2007, appendix A provides an in depth review of algebraic structures, including finite fields.

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

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

[11] 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.

[12] O. Trullols-Cruces, J. Barcelo-Ordinas, and M. Fiore, "Exact decoding probability under random linear network coding," *Communications Letters, IEEE*, vol. 15, no. 1, pp. 67 –69, january 2011.

[13] X. Zhao, "Notes on "exact decoding probability under random linear network coding"," *Communications Letters, IEEE*, vol. 16, no. 5, pp. 720 –721, may 2012.

[14] 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.

[15] H. Shojania and B. Li, "Parallelized progressive network coding with hardware acceleration," in *Quality of Service, 2007 Fifteenth IEEE International Workshop on*, June 2007, pp. 47–55.

[16] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, "Cautious view on network coding - from theory to practice," *Journal of Communications and Networks (JCN)*, vol. 10, no. 4, pp. 403–411, December 2008.

[17] X. Chu, K. Zhao, and M. Wang, "Massively parallel network coding on gpus," in *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, December 2008, pp. 144–151.

[18] H. Shojania and B. Li, "Random network coding on the iphone: fact or fiction?" in *NOSSDAV '09: Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*. ACM, June 2009, pp. 37–42.

[19] H. feng (Francis) Lu, "Binary linear network codes," in *IEEE Information Theory Workshop on Information Theory for Wireless Networks*, July 2007.

[20] X. Li, W. H. Mow, and F.-L. Tsang, "Singularity probability analysis for sparse random linear network coding." in *IEEE International Conference on Communications (ICC)*, June 2011.

[21] 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.

[22] E. Erez and M. Feder, "Convolutional network codes," in *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*. IEEE, Jun. 2004, pp. 146+.

[23] S. Y. R. Li and R. W. Yeung, "On convolutional network coding," in *Information Theory, 2006 IEEE International Symposium on*. IEEE, Jul. 2006, pp. 1743–1747.

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

[25] P. Maymounkov. (2006) Perpetual codes: cache-friendly coding. Unpublished draft, retieved 2nd of September 2011. [Online]. Available: http://pdos.csail.mit.edu/~petar/papers/maymounkov-perpetual.ps

[26] C. Fragouli and E. Soljanin, "A connection between network coding and convolutional codes," in *IEEE International Conference on Communications (ICC)*, 2004.

[27] S. Jaggi, M. Effros, T. Ho, and M. Mdard, "On linear network coding," in *42st Annu. Allerton Conf. Communication Control and Computing*, 2004.

[28] 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.

[29] 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.

[30] (2012) Perpetual code implementation. Currently not publically available, but will be at time of publication. [Online]. Available: https://github.com/steinwurf/kodo/tree/master/kodo/perpetual