On Tunable Sparse Network Coding in Commercial Devices for Networks and Filesystems

Sørensen, Chres Wiant

DOI (link to publication from Publisher):
10.5278/vbn.phd.tech.00028

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

ON TUNABLE SPARSE NETWORK CODING IN COMMERCIAL DEVICES FOR NETWORKS AND FILESYSTEMS

BY
CHRES WIANT SØRENSEN

DISSERTATION SUBMITTED 2017

AALBORG UNIVERSITY
DENMARK

On Tunable Sparse Network Coding in Commercial Devices for Networks and Filesystems

Ph.D. Dissertation

Chres Wiant Sørensen

Dissertation submitted June 21, 2017

Dissertation submitted: June 21, 2017

PhD supervisor: Assoc. Prof. Daniel E. Lucani
Aalborg University, Denmark

Assistant PhD supervisor: Prof. Muriel Médard
Massachusetts Institute of Technology, USA

Prof. Frank H. P. Fitzek
Technische Universität Dresden, Germany

PhD committee: Associate Professor Jan Østergaard (chairman)
Aalborg University, Denmark

Professor Sergio Palazzo
University of Catania, Italy

Associate Professor Dejan Vukobratović
University of Novi Sad, Serbia

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Electronic Systems

ISSN (online): 2446-1628
ISBN (online): 978-87-7112-982-3

Published by:
Aalborg University Press
Skjernvej 4A, 2nd floor
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2017

Abstract

Network engineers and researchers are faced with new challenges in developing upcoming 5G networks. Beyond higher throughput in broadband services, ultra-reliability, low latency and massively scalable architectures are required for emerging services, e.g., Internet of Things (IoT), Industry 4.0. To address these requirements on an increasingly wireless and mobility dependent Internet infrastructure, novel paradigms for networking, e.g., network coding (NC), are required. Network Coding (NC) revolutionizes the way communication networks operate, due in part to its rateless property, i.e., the ability to generate a virtually limitless stream of data, with minimal or no acknowledgements/signaling needed from the receivers. This entails an added cost of coding (processing) in end devices and intermediate network nodes. A computational cost that has been one of the practical limitations to NC's widespread deployment.

Tunable Sparse Network Coding (TSNC) is a NC scheme that permits sources to trade-off computational effort in end devices at the expense of coded packets that become less likely to be innovative, i.e., provide new knowledge to receivers. This thesis builds on early ideas of TSNC and delivers various practically relevant mechanisms for its deployment in real systems. First, this thesis presents a practical implementation of TSNC that applies a limited number of feedback packets to allow a source to monitor and regularly tune the trade-off between computational complexity and the transmission delay. This scheme is benchmarked in a variety of devices ranging from embedded devices, smartphones and computers. The measurements show order of magnitude performance gains of coding using TSNC compared to Random Linear Network Coding (RLNC), and a 4x to 18x speed-up using Single Instruction Multiple Data (SIMD). Second, we establish a connection between TSNC and techniques like Overlapping Generations (OG) by providing an on-the-fly coding construction of OG. More specifically, we use feedback packets to construct the overlapping generations and sparse coding to maintain low complexity. We show that this can be seen as a TSNC approach, where we tune the innovation probability of coded packets based on the feedback received. This scheme is compared to three other NC schemes.

Finally, this thesis presents a new paradigm in the form of a filesystem protocol. We use NC at the filesystem level of end devices, i.e., on the top of the network stack, to expand regular and even proprietary protocols with the benefits from coding. This mechanism is realized into a real-life proof-of-concept implementation based on the Filesystem in Userspace (FUSE) library to provide multipath multi-source downloading capabilities for legacy protocols, such as the Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP). We show by measurements how our Network Coded Filesystem Shim (NCFSS) provides two to five fold performance gains when multiple server mirrors are used to download 10 MiB and 100 MiB files using regular HTTP connections.

Resumé

Netværksingeniører og forskere står overfor nye udfordringer med at udvikle kommende 5G netværk. Udover højere kapacitet på bredbåndsforbindelser er ultra-pålidelighed, lave forsinkelser og meget skalerbare arkitekturer en nødvendighed indenfor nye tjenester, f.eks., Tingenes internet, Industri 4.0. For at adressere disse krav i en øget trådløs og mobilitets afhængig internet infrastruktur er nye paradigmer indenfor netværk nødvendige, f.eks., netværkskodning. Netværkskodning revolutionerer måden hvorpå kommunikationsnetværk opererer, dels på grund af dets evne til at generere en stortset ubegrænset strøm af data med minimal eller uden behov for feedback signaler fra modtagere. Det indebærer en tilføjet omkostning til kodning (processering) i slutenheder og mellemliggende netværksnoder. En omkostning der udgør en af de praktiske begrænsninger for at udbrede netværkskodning.

TSNC er en kodningsteknik som tillader afsendere at reducere beregningskompleksiteten i slutenheder på bekostning af kodet pakker der er mindre innovative, dvs., give ny viden til modtagere. Denne afhandling bygger på tidlige ideer fra TSNC og giver forskellige praktisk relevante mekanismer til dets implementering i virkelige systemer. Først præsenterer denne afhandling en implementation af TSNC, der benytter et begrænset antal feedback pakker til at give afsenderen mulighed for løbende at monitorere og justere forholdet mellem processeringskompleksitet og transmissionsforsinkelse. Denne kodningsteknik benchmarkes i en række forskellige enheder fra inlejerede systemer til smartphones og computere. Målingerne viser omkring ti gange hurtigere kodningshastigheder med TSNC i forhold til RLNC, og en 4x til 18x højere hastighed med SIMD. Derudover etablerer vi en forbindelse mellem TSNC og teknikker såsom overlappende generationer (OG) i form af en kodningskonstruktion hvor generationer konstrueres løbende. Mere specifikt bruger vi feedback pakker til at konstruere overlappende generationer og sparsom kodning til at vedligeholde en lav kodningskompleksitet. Vi viser at metoden kan betragtes som TSNC, hvor vi justere innovationssandsynligheden af kodet pakker baseret på modtaget feedbacks. Denne kodningsteknik sammenlignes med tre andre kodningsteknikker.

Til sidst introducerer vi et nyt paradigme i form af en filsystemsprotokol. Vi implementerer netværkskodning i filsystemet på slutenheder, dvs., på toppen af netværksstakken, til at udvide almindelige og endda proprietære protokoller med fordelene ved kodning. Mekanismen bliver realiseret i form af en praktisk implementering baseret på FUSE biblioteket til at give klassiske protokoller, såsom HTTP og FTP, mulighed for at hente en fil over flere forbindelser og afsendere samtidig. Vi viser med målinger hvordan vores netværkskodningslag i filsystemet giver to til fire gange hurtigere downloads af 10 MiB og 100 MiB filer ved brug af almindelig HTTP forbindelser til at modtage data fra flere server spejle.

Contents

Thesis Details

Thesis Title:	On Tunable Sparse Network Coding in Commercial Devices for Networks and Filesystems
Ph.D. Student:	Chres Wiant Sørensen
Supervisors:	Assoc. Prof. Daniel E. Lucani Aalborg University, Denmark
	Prof. Muriel Médard Massachusetts Institute of Technology, USA
	Prof. Frank H. P. Fitzek Technische Universität Dresden, Germany

The main body of this thesis consist of the following papers.

[A] Chres W. Sørensen, Arash S. Badr, Juan A. Cabrera, Daniel E. Lucani, Janus Heide, Frank H. P. Fitzek, "A practical view on tunable sparse network coding," *European Wireless (EW)*, 2015.

[B] Chres W. Sørensen, Achuthan Paramanathan, Juan A. Cabrera, Morten V. Pedersen, Daniel E. Lucani, Frank H. P. Fitzek, "Leaner and meaner: Network coding in SIMD enabled commercial devices," *IEEE Wireless Communications and Networking Conference (WCNC)*, 2016.

[C] Chres W. Sørensen, Daniel E. Lucani, Frank H. P. Fitzek, Muriel Médard, "On-the-fly overlapping of sparse generations: A tunable sparse network coding perspective," *IEEE Vehicular Technology Conference (VTC)*, 2014.

[D] Chres W. Sørensen, Daniel E. Lucani, Muriel Médard, "On network coded filesystem Shim: Over-the-top multipath multi-source made easy," *IEEE International Conference on Communications (ICC)*, Accepted, 2017.

In addition to the main papers, the following publications have also been made.

[1] Morten V. Pedersen, Daniel E. Lucani, Frank H. P. Fitzek, Chres W. Sørensen and Arash S. Badr, "Network coding designs suited for the real world: What works, what doesn't, what's promising," *IEEE Information Theory Workshop (ITW)*, 2013.

[2] Soheil Feizi, Daniel E. Lucani, Chres W. Sørensen, Ali Makhdoumi, Muriel Médard, "Tunable sparse network coding for multicast networks," *International Symposium on Network Coding (NetCod)*, 2014.

[3] Daniel E. Lucani, Morten V. Pedersen, Diego Ruano, Chres W. Sørensen, Frank H. P. Fitzek, Janus Heide, Olav Geil, "Fulcrum network codes: A code for fluid allocation of complexity," 2015.

[4] Chres W. Sørensen, Néstor J. Hernández Marcano, Juan A. Cabrera Guerrero, Simon Wunderlich, Daniel E. Lucani, Frank H. P. Fitzek, "Easy as pi: A network coding raspberry pi testbed," *Electronics*, vol. 5, no. 4, 2016.

[5] Néstor J. Hernández Marcano, Chres W. Sørensen, Juan A. Cabrera G., Simon Wunderlich, Daniel E. Lucani, Frank H. P. Fitzek, "On goodput and energy measurements of network coding schemes in the raspberry pi," *Electronics*, vol. 5, no. 4, 2016.

[6] Pablo Garrido, Chres W. Sørensen, Daniel E. Lucani, Ramon Aguero, "Performance and complexity of tunable sparse network coding with gradual growing tuning functions over wireless networks," *IEEE International Symposium Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2016.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The thesis is not in its present form acceptable for open publication but only in limited and closed circulation as copyright may not be ensured.

Preface

This PhD thesis compose a selection of papers to represent the research made over a period of three years as a PhD student in the Antennas, Propagation and radio Networking (APNet) and Wireless Communication Networks (WCN) sections in the Department of Electronic Systems at Aalborg University. The work was done under the supervision of Daniel E. Lucani, Muriel Médard and Frank H. P. Fitzek. It was financed by the TuneSCode Project (grant dff – 1335-00125) granted by the Danish Council for Independent Research.

Acknowledgements

A special thanks to my supervisors (Daniel E. Lucani, Muriel Médard and Frank H. P. Fitzek) who made this PhD project possible and who believed in my ability to work on the TuneSCode project for three years. I appreciate all the valuable feedback, guidance and assistance throughout my entire PhD programme, and the opportunity to be a part of their research teams.

I would also like to thank the staff in APNet and WCN at Aalborg University and the Research Laboratory of Electronics (RLE) at Massachusetts Institute of Technology (MIT) for hosting me and always being available to assist me. Thanks to my colleagues and friends at Aalborg University and MIT for making the long hours at work interesting, joyful and educational. It has been a pleasure to be a part of the teams. I enjoyed all our discussions, collaboration and social activities. Finally, I would like to thank my family and friends for all the support during the recent years. I am looking forward to spend more time with all of you.

Chres Wiant Sørensen
Aalborg University, June 21, 2017

Acronyms

APNet Antennas, Propagation and radio Networking. xi

BP Belief Propagation. 17, 18, 21

Btrfs B-tree file system. 93

cp Copy. 28, 92

CPU Central Processing Unit. 21, 27, 31, 100

DOF Degree of Freedom. 9, 11–14, 16, 17, 20, 24–26, 29, 38

EXT Extended Filesystem. 93

FTP File Transfer Protocol. iv, vi, 28

FUSE Filesystem in Userspace. iv, vi, 34, 38, 92–94, 99, 104

GF Galois Field. 8, 10, 12, 13, 15, 17, 20, 26, 27, 97

GmailFS GMAIL Filesystem. 94

HTTP Hypertext Transfer Protocol. iv, vi, 28, 34, 35, 38, 95, 102–105

IoT Internet of Things. iii, 38

LDPC Low-density parity-check. 20

LT Luby Transform. 8, 17, 20, 21, 38

MIT Massachusetts Institute of Technology. xi

MPTCP Multipath TCP. 92

MTU Maximum Transmission Unit. 15

NC Network Coding. iii, iv, 7–9, 11, 13, 15, 17, 20, 23, 26–28, 31, 32, 34, 37, 38, 91–93, 95, 99, 101

NCFSS Network Coded Filesystem Shim. iv, 28, 92–94, 97–102

NFS Network File System. 93

NOG Non-overlapping Generations. 16, 17, 34

OG Overlapping Generations. iii, 17, 23, 25–27, 38

OS Operating System. 94, 99, 104

procfs Proc Filesystem. 100

QUIC Quick UDP Internet Connections. 92, 93

RAM Random Access Memory. 101

RLE Research Laboratory of Electronics. xi

RLNC Random Linear Network Coding. iii, v, 8, 9, 11–16, 20, 23–29, 31–34, 37, 38, 97, 98, 100, 105

RS Reed-Solomon. 20

SCP Secure Copy. 28, 92

SCTP Stream Control Transmission Protocol. 93

SFTP SSH File Transfer Protocol. 34

SIMD Single Instruction Multiple Data. iii, v, 27, 32, 37

SRLNC Sparse Random Linear Network Coding. 12–14, 16, 23, 24, 26, 27, 31–34, 37

SSH Secure Shell. 104

SSHFS SSH Filesystem. 94

TCP Transmission Control Protocol. 91–93

TSNC Tunable Sparse Network Coding. iii, v, 8, 13, 16, 23–27, 31–33, 37, 38

UDP User Datagram Protocol. 92

VFS Virtual Filesystem. 93, 94, 99

WCN Wireless Communication Networks. xi

Wget GNU Wget. 28, 34, 92

1 Introduction

Historically, Internet services have followed a centralized client-server paradigm, but this will likely change towards a more distributed paradigm as the number of Internet connected devices continue to increase. The declining storage prices has enabled companies to store yet more and more extensive amounts of data, i.e., big data, that even future networks, such as 5G, will struggle to support unless networks are utilized in smarter and more efficient manners than today. For example, allowing devices to share data directly among each other instead of relaying strictly on centralized servers to connect end-devices. Figure 1.1 illustrates two end devices that are connected through a remove host, indicated by the red line, to exchange information along a path that is limited in bandwidth, latency and reliability by all the intermediate nodes. This might be avoided by establishing a direct path, indicated by the dashed blue line, to provide a better link between the end devices, that also reduces the overall load on the network infrastructure.

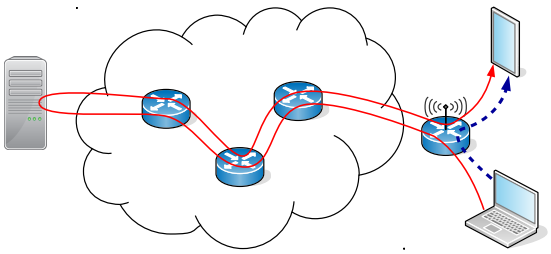


Fig. 1.1: Communication through a relay server versus peer-to-peer communication.

Communicating on more direct paths is not only beneficial in terms of bandwidth, latency and reliability, but it also opens the possibility of utilizing networks more efficiently, e.g., using multicast, broadcast and more advanced mechanisms to transmit data, e.g., NC. Compared to traditional communication standards, NC drastically changes the way data is disseminated on erasure channels due to its rateless property and reduced need for acknowledgements. This comes at the inconvenience of encoding and decoding data in end devices, and optionally intermediate network nodes, which require an additional computational effort. This thesis consider means to

which NC can be used in resource efficient manners for practical applications, e.g., using TSNC and other sparse network codes.

Chapter 1 will provide the background and state of the art on existing NC concepts and schemes, discuss their benefits and challenges, as well as discussing related sparse coding techniques such as fountain codes, e.g., Luby Transform (LT) codes, Raptor codes. Chapter 2 provides a summarized description of our work beyond state of the art. Finally Chapter 3 describes the specific contributions of the articles submitted as part of this Thesis as a collection of papers.

1.1 Random Linear Network Coding

Network coding was proposed in 2000 by Ahlswede et al. [2]. Since then, it has been suggested as a means to improve many aspects on the transportation of network packets. Network codes are rateless, meaning that a source can potentially generate an unlimited number of coded packets $C_i, i \in [1, 2, \ldots]$ from a finite set of original packets

$$P_j, j \in [1, 2, \ldots, n], \tag{1.1}$$

called a generation. This is particularly useful on erasure channels, because it does not matter which coded packets a sink receives as long as it collects n linearly independent coded packets. This means that it does not matter which coded packets are lost on the channel because the following packets are equally likely to be innovative, i.e., provide new knowledge to a receiver. In contrast, current networks require a sink (receiver) to collect every original packet (arranged in the correct order) in order to recover the original data. This ordering of the packets is performed automatically using RLNC when coded packets are decoded.

A coded packet C_i is constructed by linearly combining the original packets as

$$C_i = \bigoplus_{j=1}^{n} v_{ij} \otimes P_j, \ \forall i \in [1, 2, \ldots), \tag{1.2}$$

where v_{ij} are called the coding coefficients. For RLNC, the v_{ij}'s are chosen uniformly at random from a Galois Field (GF) of size q, and the mathematical operations are preformed over GF arithmetics. The idea of transmitting coded packets instead of the regular packets completely changes the concept of data transmissions that has historically relied primarily on per packet acknowledgements to initiate re-transmissions of lost packets. Coding the packets enables a source to construct q^n different coded packets from a generation, where all coded packets are likely to be innovative. This means that

a source is encouraged to always transmit newly generated coded packets rather than re-transmitting lost packets. Hence, eliminating the demand for per packet acknowledgements. The work in [17] allows us to calculate the probability that a coded packet is linearly independent after a receiver has collected i out of n Degree of Freedom (DOF) as

$$P(n,i) = 1 - \left(\frac{1}{q}\right)^{n-i}. \tag{1.3}$$

Using this innovation probability, we can estimate the number of coded packets B that a receiver on average needs to receive to recover a generation of n packets as

$$B = \sum_{i=0}^{n-1} \frac{1}{P(n,i)}. \tag{1.4}$$

Calculating B for various generations sizes, reveals that the overhead due to linearly dependent packets becomes insignificant if n and/or q are large

$$B \approx \begin{cases} n + 1.6 & \text{for } q = 2 \\ n & \text{for } q \geq 256 \end{cases}.$$

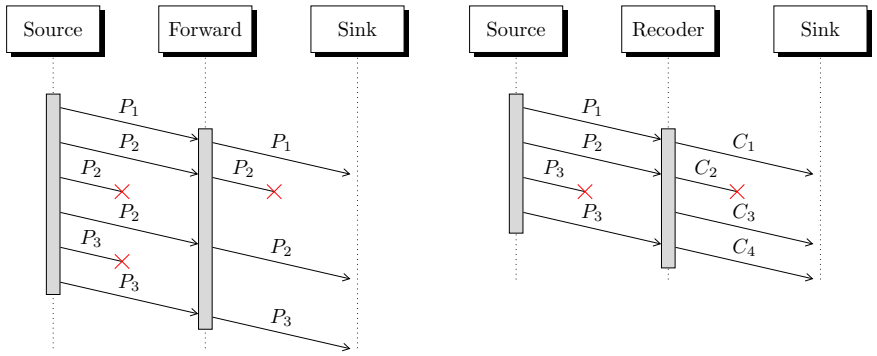
We will be using these two field sizes extensively throughout this thesis.

When coded packets arrive at a receiver, this receiver is capable of decoding the packets on-the-fly using Gauss-Jordan elimination [11]. This process requires a receiver to be aware of how each coded packet was constructed at the source, i.e., which original packets made up a coded packet. There exist two common ways to disseminate that knowledge from source to sink. Either the coding coefficients $v_i = \left[v_{i1}, v_{i_2}, \ldots, v_{in}\right]$ are transmitted along the coded packet C_i, or by the transmission of a seed associated to a pseudo-random number generator used to generate the coding coefficients on the receiver.

Both encoding and decoding carries a certain computational effort. This means that NC should only be applied when the benefits of coding outweigh the additional computational effort. This effort is high for RLNC, but one may take advantage of NC in less computationally expensive manners, e.g., systematic coding [14, 27] and sparser NC schemes. In the following, we will consider some of the benefits of NC as well as means to reduce the coding complexity.

1.1.1 Recoding

Recoding (or re-encoding) is one of the main benefits of NC. It allows intermediate network nodes to take any number of packets (either coded or not) from a generation and encode them together without having to decode them



(a) Store-and-forward **(b)** Store-and-recode

Fig. 1.2: Sequence diagrams of a transmission of three packets without recoding (left) and with recoding (right) in the intermediate node on erasure channels

first [13]. The recoding process is essentially similar to the encoding process except that recoding is usually performed over the entire content of a coded packet, i.e., both coding coefficients and codeword. The recoding process may be expressed as

$$R_i = \bigoplus_{j=1}^{n} w_{ij} \otimes C_j, \ \forall i \in [1, \ldots), \tag{1.5}$$

where w_{ij} $\forall j \in [1, 2, \ldots, n]$ forms the coding vector used to construct the i-th recoded packet R_i. Each value within the vector of coding coefficients is chosen uniformly at random from a GF of size q.

The concept of recoding is visualized in Figure 1.2, where it is compared to the familiar store-and-forward paradigm used in the Internet today. In the example, a source transmits a generation of three packets to a sink over erasure channels through an intermediate network node that either follow the store-and-forward paradigm or the store-and-recode paradigm in the two sequence diagrams respectively. The assumption is that acknowledgements are received without erasures and delays from all nodes that receive regular packets, and that only a single acknowledgement is transmitted from the sink receiving coded packets to indicate the reception of the entire generation.

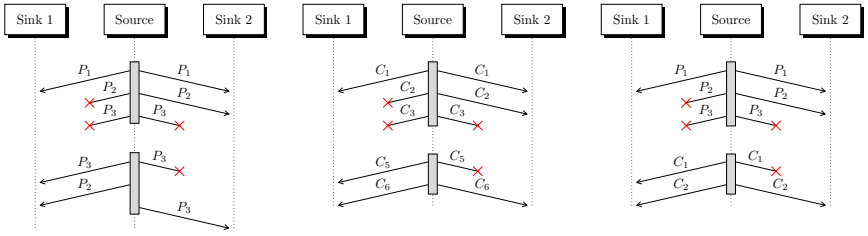
Figure 1.2 provides an idea of the store-and-recode paradigm's superior performance compared to store-and-forward. In fact, it has been proven that the probability of a sink to receive a packet transmitted over a chain of recoding nodes on erasure channels is the erasure probability of only the worst link

$$p_{\text{recv}} = 1 - \max\{e_1, e_2, \ldots, e_J\}, \tag{1.6}$$

where $e_1, e_2, \ldots, e_J$ represents the erasure probabilities of each intermediate link. Thus, far superior to the regular store-and-forward paradigm on which the probability of receiving a packet can be calculated as

$$p_{\text{recv}} = \prod_{j=1}^{J} (1 - e_j). \tag{1.7}$$

1.1.2 Multicast Performance



(a) Uncoded (per packet feedback)

(b) RLNC (end feedback)

(c) Systematic RLNC (end feedback)

Fig. 1.3: Multicast transmission of $n = 3$ packets. Uncoded multicast (using per packet acknowledgements) compared to coded multicast (using per generation acknowledgements).

Another benefit of NC is the increased throughput in multicast scenarios compared to traditional data transmissions. Transmitting packets reliably to a large group of receivers, on a wireless erasure channel, has historically been practically challenging, due to the extend of acknowledgements emitted from the receivers. Tailored multicast protocols mitigates this issue in various ways, e.g., probabilistic acknowledgements [18] and negative acknowledgements [1], but it does not solve the issue that packets need to be transmitted individually to each receiver to finalize the transmission.

Figure 1.3a provides an example of a source transmitting three packets reliably to two receivers over an erasure channel. Assuming reliable per packet acknowledgements to arrive without delays allows the source to monitor the transmission progress. After initially attempting to transmit all packets, the source may retransmit P_3 to both receivers, but it is unable to utilize the channel fully when re-transmitting P_2 to `Sink 1` and P_3 once again to `Sink 2` to finalize the transmission. RLNC on the other hand, in Figure 1.3b, allows the source to utilize the full capacity of the multicast channel in this example to feed both receivers with coded packets until they obtain all three DOF. As previously discussed, the coding process is not free in terms of computational effort. Figure 1.3c shows a commonly used approach to save computational

effort by first transmitting the entire generation uncoded before starting the coding process. This is called systematic RLNC [14, 27].

1.2 Sparse Random Linear Network Coding

Sparse Random Linear Network Coding (SRLNC) is a computational inexpensive alternative to RLNC. It is similar to RLNC except that each coded packet is constructed only from a subset of the regular packets in a generation. This allows an encoder to construct coded packets that are less computationally expensive to both encode and decode at the cost of reducing the probability of packets being innovative. Thus, a sink needs to receive more coded packets to obtain n linear independent packets compared to RLNC. Constructing SRLNC coded packets follow the same procedure as RLNC

$$C_i = \bigoplus_{j=1}^{n} v_{ij} \otimes P_j,\ v_{ij} = \mathcal{Z}_j,\ \forall i \in [1, \ldots), \tag{1.8}$$

but with coding coefficients sampled from a discrete distribution

$$Z_j \sim \rho(z) = \begin{cases} 1-d & \text{for } z = 0 \\ \frac{d}{q-1} & \text{for } z = 1, \ldots, q-1, \end{cases}$$

that only returns a non-zero value with probability d. This means that an encoder constructs each coded packet from $k = d \cdot n$ regular packets on average. In practical applications, we restrict the coding density to the following ranges [22]

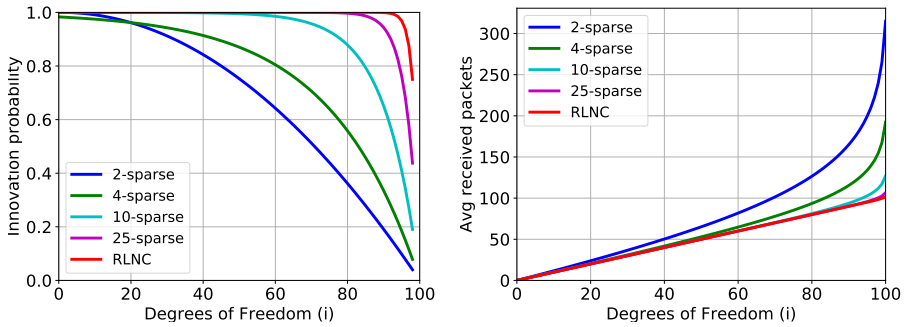
$$d \in \begin{cases} (0, 0.5] & \text{for } q = 2 \\ (0, 1] & \text{for } q > 2. \end{cases} \tag{1.9}$$

Similar to RLNC, the innovation probability depends on the number of linearly independent packets a receiver has already accumulated and the generation size, but the coding density d is much more dominant than the GF size q. We use the equation from [5] to calculate the innovation probability of a sparse coded packet after a sink has recovered i out of n DOF as

$$P(n, i) = 1 - (1-d)^{n-i}. \tag{1.10}$$

Thus, the number of packets a receiver needs to receive to recover all regular packets may be estimates as

$$B = \sum_{i=0}^{n-1} \frac{1}{P(n,i)} = \sum_{i=0}^{n-1} \frac{1}{1-(1-d)^{n-i}}. \tag{1.11}$$



(a) Innovation probability **(b)** Received packets

Fig. 1.4: SRLNC innovation probability (left) and the average received packets (right) at i DOF using GF of size $q = 2$.

The innovation probability of SRLNC coded packets are highly dependent on the coding density d. This is illustrated in Figure 1.4a for five various densities and a generation size of $n = 100$ packets. The densities are expressed as k-sparse where $k = d \cdot n$. It can be seen how the innovation probability decays as the receiver accumulates more DOF, and that the innovation probability is lower when d is small. These effects are also apparent if we consider the number of packets a receiver needs, on average, to recover i DOF. This is shown in Figure 1.4b using the same coding densities. The RLNC curves, in the two figures, illustrate the best achievable performance of NC using $q = 2$, where the optimal performance would be a fixed innovation probability of one and hence a DOF increase by one for each received packet.

1.3 Tunable Sparse Network Coding

TSNC [6] can be considered as an extension of SRLNC. It encodes and decodes packets similar to SRLNC, but instead of a fixed coding density throughout the transmission of each generation, TSNC suggests to increase the coding density d as the receiver(s) accumulates more linearly independent packets from a generation. This enables a source to reduce coding complexity in an intelligent fashion compared to SRLNC that lose particularly many packets, due to linearly dependency, when receivers have accumulated a large number of DOF. Figure 1.4a illustrated 1) how sparse coded packets are almost guaranteed to be innovative in the beginning of a transmission no matter the coding density, and 2) how the innovation probability decays slower for denser codes compared to sparser codes when receiver(s) accumulates more DOF.

TSNC enables a source to reduce coding complexity at only a small addi-

tional coding delay compared to RLNC by initially transmitting very sparse coded packets and gradually increasing the coding density as the receiver(s) accumulate DOF. Because the coded packets are sparse, the innovation probability is similar to that of SRLNC in Equation 1.10. This means that the average overhead $g(i)$ of receiving a linear independent coded packet when a receiver has accumulated i out of n DOF may be expressed as

$$g(i) = \frac{1}{P(n,i)} - 1 = \frac{1}{1-(1-d(i))^{n-i}} - 1. \tag{1.12}$$

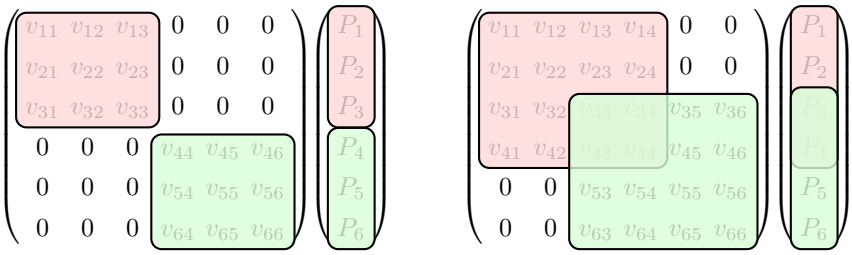
The mean number of packets to be received by a receiver to decode a generation may be calculated using the same equation presented for SRLNC. The expression may be written as n linearly independent packets plus the accumulated overhead as

$$B = \sum_{i=0}^{n-1} \frac{1}{P(n,i)} = n + \sum_{i=0}^{n-1} g(i). \tag{1.13}$$

This expression simplifies how a source can control the overhead $g(i), i \in [0, 1, \ldots, n-1]$ that may be constructed as a discrete function to suit any overhead behaviour for a given application. Knowing the overhead function $g(i)$ allows us to calculate the i-th coding density.

$$d(i) = 1 - \left(1 - \frac{1}{1+g(i)}\right)^{\frac{1}{n-i}}. \tag{1.14}$$

1.4 Non-Overlapping Generations



(a) Non-overlapping generations **(b)** Overlapping generations

Fig. 1.5: Non-overlapping generations and overlapping generations

Non-overlapping generations [4, 20] is another way to reduce the coding complexity of NC. RLNC enables the transmission of generations in $n + 1.6$ coded packets in the expectation if $q = 2$, and near n coded packets for larger GF sizes. Regardless of the generation size, there will always be roughly the same number of linear dependent packets, e.g., 1.6 using $q = 2$. This means that the overhead due to linearly dependent packets is less significant for larger generations than for small generations. On the other hand, the coding complexity is affected negatively as the generation size grows. In fact, this is a key limitation of RLNC since Gauss-Jordan elimination requires $O(n^3)$ GF operations to decode a generation of n packets. RLNC based protocols are therefore forced to use either very large packets or splitting the packets into multiple generations that can be transmitted one by one. Due to the Maximum Transmission Unit (MTU) and other system limitations, the solution is often limited to the use of multiple generations, e.g., as illustrated in Figure 1.5a, where the matrix elements represent the coding coefficients and the vector elements represent the original data packets.

Splitting the packets into h generations of sizes $[m_1, \ldots, m_h]$ reduces the decoding complexity to $O\left(\sum_{j=1}^{h} m_j^3\right)$ GF operations at the cost of transmitting more linearly dependent packets. The mean number of packets a receiver needs to collect to decode h generations is

$$B = \sum_{j=1}^{h} \sum_{i=0}^{m_j - 1} \frac{1}{P(m_j, i)}. \tag{1.15}$$

This means that a receiver on average needs to collect $\sum_{j=1}^{h}(m_j + 1.6)$ packets if all generations are RLNC encoded with a GF of size $q = 2$.

The generations may be encoded using other schemes such as SRLNC or TSNC. In that case, the coding complexity is reduced even more, but at a cost of more linearly dependent packets.

This is illustrated in Figure 1.6 that shows the innovation probabilities of coded packets as a receiver accumulates DOF. A generation of 100 original packets are split into three non-overlapping generations that are received sequentially either using 2-sparse or 4-sparse coded packets. As expected, the innovation probabilities of received packets are high in the beginning of each generation, but decays as more DOF are accumulated. We observe that the 4-sparse coded packets are always more innovative than 2-sparse coded packets at the expense of higher coding complexity.

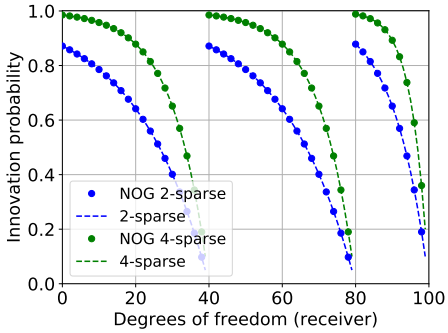


Fig. 1.6: Innovation probability of coded packets using three Non-overlapping Generations (NOG) of sizes $[40, 40, 20]$ when receiving a total of 100 original packets.

1.5 Overlapping Generations

The concept of overlapping generations [10, 26, 28] is usually applied in transmissions of large files as an alternative to non-overlapping generations. The difference between the two are that regular packets may be included in multiple generations in overlapping generations. Thus, when a packet is fully decoded in one of the generations, it can be eliminated from all other generations that includes it. Figure 1.5 provides an illustration of non-overlapping and overlapping generations for a total of six packets that are split into two generations. From the coding matrix, it can be seen that some coding coefficients are always zero. Thus, compared to a single large RLNC generation, the packets are less computationally expensive to encode and decode when split into multiple generations.

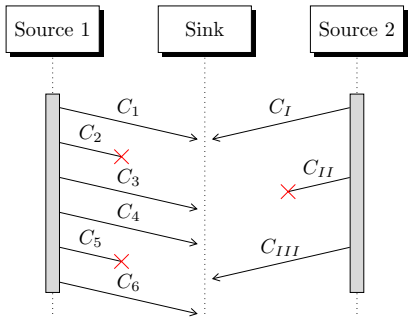


Fig. 1.7: Sequence diagram of retrieving a single data chunk from multiple sources using OG

Figure 1.7 visualizes of how overlapping generations can be beneficial to receive the two generations in Figure 1.5b. One generation is transmitted from each source. $\{P_1, \ldots P_4\}$ from `Source 1` and $\{P_3, \ldots, P_6\}$ from `Source 2`. Due to the overlap, when the sink receives the fourth DOF from `Source 1`, it can decode the first generation and thus use P_3 and P_4 to decode the other generation that was transmitted by the slower `Source 2`. Had the two generations been transmitted using non-overlapping generation, such as in Figure 1.5a, then it would be required that the sink received the entire generations from both sources. Although the generations are bigger due to overlaps, the coding effort may turn out lower compared to NOG. This happens because a fully decoded packet in one generation may be eliminated in all generations that it belongs to with little effort.

1.6 Luby transform (LT) and Raptor codes

LT [15] and Raptor [25] codes make up two low complexity competitors to NC. Both codes are rateless, thus enabling encoders to potentially produce a limitless stream of coded packets $C_i, i \in [1, 2, \ldots]$ from a finite set of original packets $P_j, j \in [1, 2, \ldots, n]$.

To minimize coding complexity, both codes relies on the Belief Propagation (BP) decoder [16] to recover the original packets, i.e., a low computationally expensive alternative to Gauss-Jordan elimination, that works exclusively on the binary GF, i.e., $q = 2$. The coding operations of BP can best be explained using a bipartite graph, such as Figure 1.8, that depicts three original packets (P_1, P_2, P_3) and four coded packets $(C_1, \ldots, C_4)$.

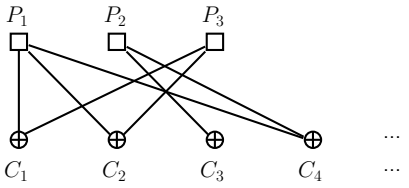


Fig. 1.8: LT code

Each coded packet C_i is produced using three steps, 1) uniformly at random choose a degree ω from a degree distribution, 2) select ω distinct packets from $P_j, j \in [1, 2, \ldots, n]$ randomly as neighbors to C_i, and 3) construct C_i by XORing its ω neighbors.

Decoding coded packets is performed in steps iteratively. First, coded packets with exactly one neighbor represent exact copies of their neighboring original packet. These packets are marked as recovered. Second, the edges from the newly recovered packets to coded packets are eliminated using bitwise XOR to decrease the degree of the affected coded packets by one. Third, the steps are repeated until all original packets are recovered.

The fact that BP demands coded packets of degree-one to start the decoding process and keep it running by eliminating edges makes it vulnerable to the coding density. In fact, it works only on very sparse code structures and a good performance, in terms of coding delay, is only achieved using carefully tailored degree distributions, such as the Ideal Soliton distribution

$$\rho(i) = \begin{cases} \frac{1}{n} & \text{for } i = 1 \\ \frac{1}{i\,(i-1)} & \text{for } i = 2, \ldots, n. \end{cases} \tag{1.16}$$

The distribution has been designed such that each newly recovered packet is eliminated from roughly $R = 1$ coded packet in each iteration of the decoding process. This works great in theory, but it performs poorly in practice. Even small fluctuations from the expected behaviour cause the decoding process to halt due to missing degree-one packets. This problem was addressed in the Robust Soliton distribution by altering the Ideal Soliton distribution such that each recovered packet roughly eliminates one degree from

$$R = c \, \ln\left(\frac{k}{\delta}\right) \sqrt{k}$$

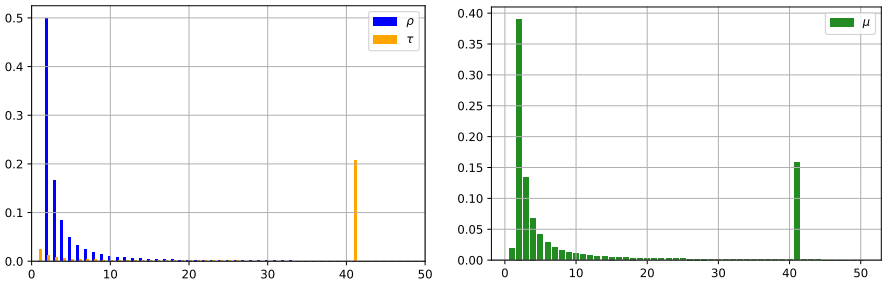
coded packets. Thus, making the decoding process less likely to halt due to missing degree-one packets. c is a constant larger than zero and δ is the probability of failure to recover all original packets after the reception of B coded packets.

The Robust Soliton degree distribution is defined as

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\beta} \quad \text{for } i = 1, \ldots, n, \tag{1.17}$$

where $\beta = \sum_{i=1}^{n}(\rho(i) + \tau(i))$ and

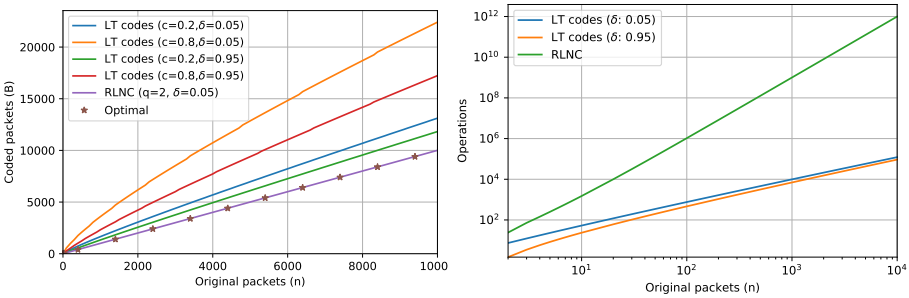
$$\tau(i) = \begin{cases} \frac{R}{i\,n} & \text{for } i = 1, \ldots, \frac{n}{R} - 1 \\ \frac{R \ln\left(\frac{R}{\delta}\right)}{n} & \text{for } i = \frac{n}{R} \\ 0 & \text{for } i = \frac{n}{R} + 1, \ldots, n. \end{cases} \tag{1.18}$$



(a) Ideal Soliton **(b)** Robust Soliton

Fig. 1.9: $c = 0.2, \delta = 0.05, R =$

The Ideal Soliton distribution $\rho(\cdot)$, $tau(\cdot)$ and the Robust Soliton distribution $\mu(\cdot)$ has been plotted in Figure 1.9 for an example with $n = 10000$ original packets and parameters $c = 0.2$ and $\delta = 0.05$. The plots show how $\tau(\cdot)$ significantly increase the probability of generating coded packets of degree-one to accelerate the start of the decoding process and n/R to make denser coded packets that are more likely to maintain the decoding process.



(a) Delay

(b) Decoding cost

Fig. 1.10: LT codes compared to RLNC in terms of coding delay and computational complexity

The number of coded packets required to recover a generation of n packets can be estimated as [19]

$$B = n\,\beta.$$

Figure 1.10a compares this coding delay to a worst case of RLNC using the binary GF and a 95% probability of successfully decoding a generation after receiving B coded packets [16]. Although RLNC is plotted using the binary GF it is close to optimal in terms of coding delay, but its computational complexity is significantly higher. This is illustrated in Figure 1.10b. The coding operations of LT codes is $O(n \cdot \ln(n/\delta))$ compared to $O(n^3)$ in RLNC.

This performance of LT codes can be improved using techniques suggested in Raptor codes. Raptor codes extends the ideas of LT codes with one or multiple layers of precodes as illustrated in Figure 1.11.

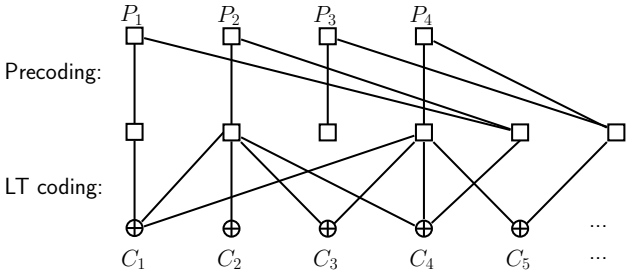


Fig. 1.11: Raptor codes

The delay of LT codes are conceptually caused by the same underlying problem as NC schemes experience of various degrees, i.e., the fact that coded packets become less innovative as receivers accumulate more DOF. The precoding in Raptor codes expands the original n packets to $n + r$ packets using a strong code, e.g., Low-density parity-check (LDPC) [8] or Reed-Solomon

(RS) codes. The trick is that Raptor decoders may therefore recover a generation with close to n out of $n + r$ packets rather than receiving the exact n original packets. The other advantage of extending the original n packets to $n + r$ packets is that decoding complexity of Raptor codes may be reduced to $O(n)$ operations. Although the decoding procedure in a Raptor code depends on the precode(s), it is most common to use BP in the entire decoding procedure to recover the original packets.

LT and Raptor codes are very attractive in many unicast and multicast scenarios where channel erasures are present. This is due to their rateless property, low coding delay and computational complexity. Compared to Gauss-Jordan Elimination, BP is theoretically superior to Gauss-Jordan elimination in terms of theoretical complexity, but it is vulnerable to the carefully tailored degree distributions to work efficiently. In practical implementations, Gauss-Jordan has proved very computationally efficient [11] due to the simple, consistent and repetitive tasks that Central Processing Unit (CPU)s perform extremely well. BP is yet to be compared to Gauss-Jordan elimination in a real-life implementation since only highly efficient proprietary implementation exists of BP. The carefully tailored degree distribution required for BP to work is also making it nearly impossible to recode LT or Raptor coded packet streams at intermediate nodes [24]. Thus, LT and Raptor codes are not able to utilize the benefits of recoding.

2 Thesis Outline

A number of papers have demonstrated benefits of applying NC in practical applications, but we are yet to see a wide scale assimilation. The concept is still relatively young, and it may not be common knowledge among system developers that would need a large technical understanding of NC to use it. Another reason that could hinder the propagation of NC may be the missing knowledge of how it performs in everyday devices.

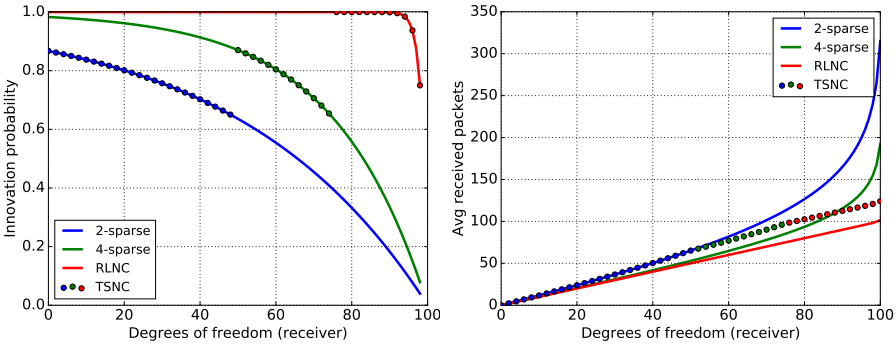
This thesis focuses on mapping the computational performance of common NC schemes in a variety of commercially available devices, how to reduce the computational effort of NC through the use of sparse code structures, such as TSNC, and how NC in general can be made easily accessible even to people without a deep technical knowledge of NC. We first considered a simple unicast network without channel erasures between a source and a sink to observe and understand the behaviour and coding performance of RLNC compared to sparser code structures. This allowed us to develop a feedback based TSNC scheme that was tested and compared to RLNC and SRLNC. We performed measurements in eight different commercially available devices, ranging from a Raspberry Pi 1 to smartphones and regular computers, to map the coding speed and energy consumption of the schemes running on various platforms.

Similar measurements were used to estimate the performance of a TSNC inspired approach to OG, proposed in Paper C. We believe this scheme may potentially be useful in multicast and multi-source applications, i.e., transmitting to multiple receivers or receiving from multiple sources. Both of which NC appears particularly interesting, due to its ability to reduce the effort and protocol overhead required to ensure that the same packet is not transmitted over multiple communication channels. In the multi-source case, this either require each source to be aware of a pre-negotiated subset of data pieces to transmit, or to continuously have the receiver(s) request data piece by piece from the sources in a BitTorrent fashion. Our final work of this thesis present ideas of how NC may be implemented as a filesystem shim in end devices to allow pre- and post-coding operations before/after any Linux application read or writes from/to its filesystem. Thus, allowing our shim to provide coding capabilites between the application layer and the filesystem, while applications may be completely oblivious to the coding performed before

transmitting and after receiving the raw data.

The work will be expanded in more details in the following sections and papers.

2.1 Feedback based Tunable Sparse Network Coding



(a) TSNC innovation probability **(b)** TSNC received packets at rank

Fig. 2.1: Innovation probability and average received packets to obtain a given number of DOF for a generation of 100 packets

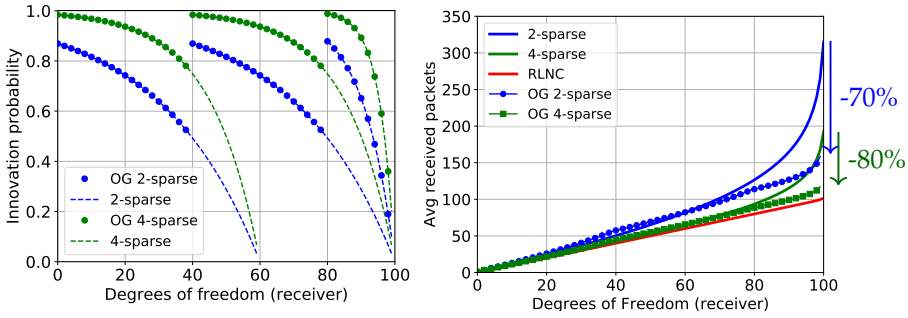
The ideas of TSNC, proposed by Feizi et. al in [6], provide a theoretical framework to trade-off computational complexity at the expense of additional delay. The delay is due to linearly dependent packets, and may be based on a desired overhead function. Papers A and B realize these ideas into a practical C++11 implementation that uses feedbacks from the receiver(s) to deliver a highly consistent and controlled delay performance to end devices. Upon the accumulation of a number of pre-negotiated degrees of freedom between the source and receiver(s), the receiver(s) transmit feedback packets to the source to indicate the current state of its decoding process. These feedbacks enable the source to regularly update the coding density and thereby transmit the entire generation to the receiver(s) in roughly $B \geq n$ transmitted packets. We suggested that receivers acknowledge the reception of each $r(k)$ DOF as

$$r(k) = \left\lfloor n \cdot \left[\frac{2^k - 1}{2^k} \right] \right\rfloor, \; k \in \Big[1, 2, \ldots, \lceil log_2(n) \rceil + 1\Big]. \tag{2.1}$$

Based on this feedback scheme, Figure 2.1 illustrates the theoretical innovation probability of coded packets, using our TSNC scheme compared to SRLNC and RLNC, when a source receives $k = 3$ acknowledgements at

$r(k) = \{50, 75, 100\}$ DOF and the coding density is 2-sparse, 4-sparse and RLNC in the three regions of a 100 packets generation. Based on the innovation probabilities, we may use Equation 1.13 to calculate the average number of coded packets a receiver needs to collect to obtain a given DOF. This is depicted in Figure 2.1b.

2.2 Feedback Based Overlapping Sparse Generations

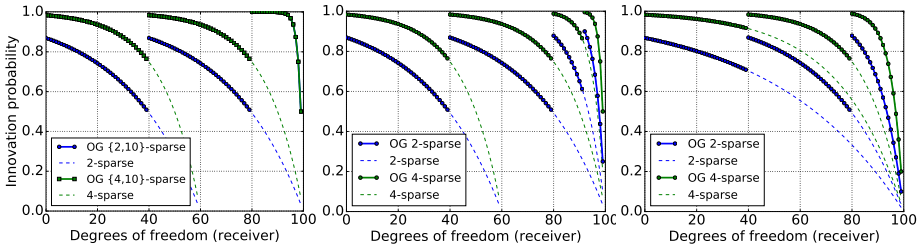


(a) Innovation probability **(b)** Received packets

Fig. 2.2: Innovation probability and average received packets to obtain a given number of DOF when receiving 40, 40 and 20 DOF from overlapping generations of sizes $[60, 60, 20]$

Our TSNC scheme, described in Section 2.1, used feedbacks to monitor and tune the coding density regularly to finish the transmission of a generation in roughly B packets. Paper C presents how these ideas may be utilized in the context of OG. Consider the transmission of a large number of packets, n, to one or multiple receivers. If n is large, it may be an advantage to transmit the packets over multiple generations to maintain a low coding complexity. Our work in Paper C suggests to construct these generations on-the-fly based on the feedback from the receiver(s). First, the source constructs an initial generation of the first m packets, from which it transmits k-sparse coded packets to the receiver(s). When the receiver(s) has recovered $m - r$ DOF of the initial generation and the innovation probability becomes significantly low, the receivers signal a list of seen or unseen packets to the source. This triggers the source to construct a new generation of similar size, that consists of the unseen packets plus a set of new packets, i.e., packets that have not been included in a previous generation. The described process may continue until all packets have been included in a generation and the very last generation has been decoded. Decoded packets in one generation can be back-substituted in all previously received generations.

Figure 2.2a presents the innovation probabilities of coded packets using our scheme with coding densities of either 2- or 4-sparse packets to recover i out of n DOF, and the generation sizes are $[60, 60, 20]$ and overlaps by $r = 20$ packets. Using these scheme configurations, Figure 2.2b, indicates a 70-80 percent reduction of the delay performance compared to their SRLNC counterpart.



(a) Denser generation **(b)** Additional generation **(c)** Larger overlap

Fig. 2.3: Innovation probability of OG with denser last generation, additional generation or larger overlap

Figure 2.2a shows that coded packets from the last generation may not be sufficiently innovative. This is due to the low coding densities. Figure 2.3 presents three ways to increase the innovation probability of our scheme while maintaining a low overall coding density. The first option could be to increase the coding density of the last generation. This is depicted in Figure 2.3a, where the very last generation is based on RLNC coded packets. In a more advanced setup, the last generation could be transmitted using TSNC or simply a GF of higher order than GF(2). Second option could be to maintain the low 2-sparse or 4-sparse coding density in all generations, but instead construct more generations. This is illustrated with four generations in Figure 2.3b. A third option could be to use larger generation overlaps as depicted in Figure 2.3c.

Although the three options presented improves the innovation probability of the last generation, they all trade off something. The first option introduces higher coding complexity, the second option adds additional feedbacks, and the third option are likely to consume more memory in a practical implementation.

2.3 Network Coding Complexity

Computational complexity is usually expressed in GF operations. These estimates provide an overall picture of the performance that may be expected of a particular NC scheme, but this measure is not necessarily proportional

to the speeds obtained in a real implementation. There exists a number of reasons that GF operations are not necessarily proportional to the computational speed in a real implementation. Our work in [22] showed that three of the most important effects are the following

- GF operations: The GF computations varies in speed depending on the operation and the GF size q. Especially high order fields usually relies on table look-ups that may be implemented in various ways depending on the system capabilities.
- CPU: The CPU frequency and capabilities varies between devices and computers are extremely good at performing repetitive and predictable tasks where they can utilize parallelism to perform multiple tasks concurrently.
- Data locality: The pattern used by the CPU to read and write data from/to memory may drastically affect the performance of an implementation. This is due to different speeds of sequential read/writes compared to random read/writes, but also the systems ability to operate on data in higher cache levels, i.e., store data closer to the CPU.

The work in Paper A evaluates the computational speed of our feedback based implementation of TSNC. Paper B compares the same TSNC scheme against RLNC and SRLNC in eight different commercially available devices, ranging from the Raspberry Pi 1 to regular computers, while the energy consumption is measured in some of the devices to map the efficiency of the measured NC schemes. Finally, Paper C estimates the performance of a TSNC inspired OG scheme compared to regular SRLNC and systematic RLNC. Due to the extremely sparse codes that may be used in TSNC, Paper B also presents a minor optimization on how a source may deduce which regular packets should form each coded packet.

From measurements, we observed performance gains in the order of magnitude using TSNC and our TSNC inspired OG scheme compared to RLNC, and 4x to 18x speed-ups using SIMD capabilities that are available in most recently available consumer CPUs.

2.4 Network Coding in Filesystems as a solution to multi-source transmissions

Practical implementations that are deployed beneath the application layer are faced with tremendous challenges to become standardized and widely accepted by the Internet community before a decade long process of updating all network nodes on the Internet can begin. For this reason, NC has mainly

been deployed on the application layer that does not require intermediate network nodes to be updated. Instead, each individual application needs to implement it, which requires a deep technical knowledge and effort to design and use properly.

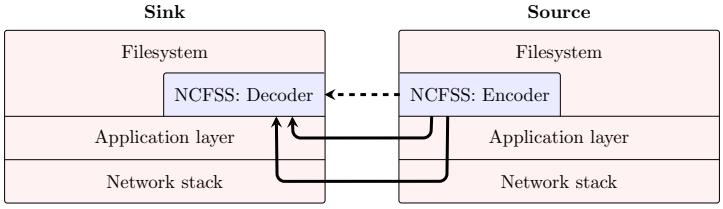
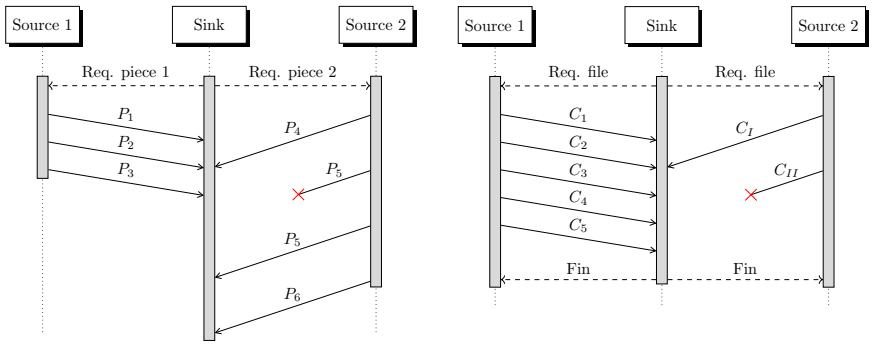


Fig. 2.4: NCFSS in the filesystem. An application may copy data directly between filesystems within the same machine (e.g. using Copy (cp)) or to a remote machine (e.g. using Secure Copy (SCP) or GNU Wget (Wget)).

In Paper D, we propose to implement NC as a shim between the filesystem and the application layer, as depicted in Figure 2.4. This allow new applications, but also existing applications that are based on legacy protocols, such as HTTP [3, 7], FTP [23], to take advantage of NC without any alteration of the existing code. This is possible due to standardized I/O interfaces between user space applications and filesystems.

Our work proposes NC as a solution to multi-source transmissions and provides a proof-of-concept implementation in C++ to demonstrate the ability and performance gains of using RLNC to download a file from multiple server mirrors.



(a) Without NCFSS **(b)** With NCFSS

Fig. 2.5: Downloading a file from multiple HTTP server mirrors

Downloading files from multiple mirrors is already feasible today, e.g., using HTTP. The simplest approach is to split a file into equally sized pieces

that a sink may request one by one from the server mirrors. Figure 2.5a illustrates a sequence diagram of a sink downloading a file, that has been split into two pieces of three packets each, such that the sink can download one piece from each mirror. This approach does not perform well in case transmission rates to the server mirrors differ.

RLNC eliminates the effort of coordinating which pieces to retrieve from each server by considering a file as a single generation from which each source may transmit a limitless stream of coded packets. This allows a sink to collect packets from multiple sources until it has retrieved enough DOF to fully reconstruct to original data. The approach is visualized in Figure 2.5b. Another benefit of coding is that it inherently provides a chunked transmission that allow faster sources to transmit more packets than slower sources. The performance of these downloading strategies are presented in Paper D.

3 Thesis Contribution

This chapter presents the contribution of each paper included in the thesis.

3.1 Paper A

Chres W. Sørensen, Arash S. Badr, Juan A. Cabrera, Daniel E. Lucani, Janus Heide, Frank H. P. Fitzek, **"A Practical View on Tunable Sparse Network Coding,"** *European Wireless (EW)*, 2015.

Motivation

TSNC was proposed in [6] as a means to obtain a lower computational complexity than RLNC by trading off coding complexity using sparse codes at the expense of increased coding delays. The work presented the potential benefits of TSNC, but it did not perform a real-life implementation of the scheme. The question is therefore if the ideas work in practice and how it performs compared to other NC schemes.

Main Content

Paper A presents a real-life implementation of a TSNC scheme that use feedbacks from receiver(s) to monitor the decoding process. The implementation is made in the Kodo C++11 library [21], and the performance of the scheme is compared against SRLNC and RLNC for various network coding parameters. The measurements are all performed in a laptop equipped with a 2.7GHz Intel Core i7-3740QM CPU.

Main Results

The measurements collected from the performance comparison of the three network coding schemes showed a performance gain of TSNC from 4x to 10x compared to RLNC. It also revealed that the estimation of the coding density could be performed with minimal performance impact. Due to the small performance impact and the imprecise innovation probabilities meant

that the optimal performance of our TSNC implementation was obtain by sending only a limited number of feedbacks.

Related Publications

We show the key enabling mechanisms for TSNC and present a new algorithm to perform Gaussian elimination efficiently on very sparse codes in Paper [5], and we expand the ideas on tuning the coding density of TSNC gradually in Paper [9].

3.2 Paper B

Chres W. Sørensen, Achuthan Paramanathan, Juan A. Cabrera, Morten V. Pedersen, Daniel E. Lucani, Frank H. P. Fitzek, **"Leaner and Meaner: Network Coding in SIMD Enabled Commercial Devices,"** *IEEE Wireless Communications and Networking Conference (WCNC)*, 2016.

Motivation

Network connected devices are becoming yet more and more diversified in terms of processing capabilities and their ability and willingness to perform computational expensive operations. It is therefore of key interest to consider network coding in various different everyday devices, such as embedded devices, smartphones and regular computers, to get an idea of the performance and energy consumption of NC. This may be useful in the pursuit of how to use NC most efficiently. The performance of NC was expected to depend not only on the coding parameters (e.g. generation size, packet sizes, field type), but also on the devices abilities to utilize hardware capabilities, such as SIMD. TSNC was of particular interest as it provides the additional means to trade-off computational complexity at the expense of additional delay.

Main Content

The paper presents the performance of RLNC, SRLNC and TSNC in eight commercially available everyday devices. These devices include Raspberry Pi, smartphones and regular computers. The performance benefits of SIMD and the energy consumption are quantified in some of the devices.

Main Results

The measurements showed a 4x to 18x improvement by using SIMD for encoding and decoding compared to not taking advantage of SIMD. It was also shown that RLNC consumed 4x to 45x more energy per encoded bit and 2.5x

to 15x per decoded bit compared to its sparser alternatives tested in the paper (SRLNC and TSNC). We observed that the energy consumption varied very little between the coding schemes, so the energy savings relied almost exclusively on the encoding/decoding speed of a given coding scheme. Thus, faster coding equals less energy per bit. The measurements were performed within each individual devices. Hence, transmission and receiving was not quantified and should be expected to have a larger negative impact on the sparse schemes that are more thrown to construct linearly dependent packets.

Related Publications

In Paper [12], we evaluate the performance of TSNC, SRLNC and RLNC in more details particularly for Raspberry Pi model 1 and Raspberry Pi model 2.

3.3 Paper C

Chres W. Sørensen, Daniel E. Lucani, Frank H. P. Fitzek, Muriel Médard, **"On-the-fly Overlapping of Sparse Generations: A Tunable Sparse Network Coding Perspective,"** *IEEE Vehicular Technology Conference (VTC)*, 2014.

Motivation

TSNC has the ability to assure that coded packets stay likely to be innovative to receiver(s) throughout the transmission of a generation by gradually increasing the coding density. Increasing the coding density contributes to more computational effort to both encode and decode coded packets. This paper proposes an alternative TSNC-like scheme to keep coded packets likely to be innovative while maintaining very sparse codes throughout the entire transmission. Instead of increasing the coding density, we propose to use feedback from the receiver(s) to eliminate already received packets from the encoder(s) transmit buffer. This approach offers TSNC like functionalities, but without increasing the coding complexity. In contrast to TSNC, it can however not work without feedbacks. It is of high interest to consider both the performance benefits of this approach and its ability to maintain the innovation probability of coded packets.

Main Content

The paper propose a new overlapping generations approach of TSNC and provides an estimate of its potential performance in terms of processing speed and innovation probabilities of coded packets. These estimates are based on practical measurements of SRLNC encoding and decoding. The

proposed scheme was compared to three other schemes. A single generation of SRLNC, multiple non-overlapping generations of SRLNC and a single generation of systematic RLNC.

Main Results

The main results of the paper showed that the proposed scheme significantly decreased the number of received packets required to decode a generation compared to SRLNC and NOG. It was seen that our scheme was very dependent on the last generation size and that larger overlaps mapped into better delay performance. The scheme also appeared to potentially reduce the processing effort by orders of magnitude.

3.4 Paper D

Chres W. Sørensen, Daniel E. Lucani, Muriel Médard, **"On Network Coded Filesystem Shim: Over-the-top Multipath Multi-Source Made Easy,"** *IEEE International Conference on Communications (ICC)*, 2017.

Motivation

Incorporating network coding into applications or protocols requires lots of effort and know-how. In fact, it may not be either possible or feasible in some cases, e.g., when applications or protocols are proprietary. The question is if NC can be incorporated around existing applications and/or protocol stacks by implementing NC as a filesystem or as a shim between applications and a regular filesystem. Thus, introducing network coding capabilities on file transmissions while relying on existing applications to handle the actual transmission.

Main Content

The work presents the benefits of placing a FUSE shim between applications and a regular filesystem. This potentially enables legacy and even proprietary applications to transmit data over multiple paths although an application might traditionally only have been developed for single flow transmissions. A proof-of-concept implementation is described and tested in Linux using Copy (cp), SSH File Transfer Protocol (SFTP) and HTTP with lighttpd as server and Wget as client.

Main Results

The paper quantified the throughput benefits of downloading large files from multiple HTTP server mirrors using network coding in contrast to regular transmission strategies. It was shown that network coding could in fact be implemented as a filesystem shim although a few caveats were introduced. The coding significantly reduced the expected downloading time compared to the traditional strategies when retrieving data from multiple HTTP server mirrors.

4 Conclusion

This thesis advocated for the use of sparse network codes as a means to benefit from the simplistic way NC solves traditionally complex issues of telecommunication, such as multicast, multipath, multi-source and packet erasures, at a low computational cost of coding in end devices and intermediate nodes.

Through a real-life implementation of a feedback based TSNC scheme, we showed how even small numbers of feedback packets were enough to drastically reduce the coding complexity compared to RLNC even at relatively small additional delay performances. This applied to the complete group of eight various devices, ranging from a Raspberry PI model 1 to smartphones and regular computers, that we used to benchmark our implementation of TSNC to SRLNC and RLNC in terms of processing speed, delay performance and the energy consumption. Our measurements showed processing gains in the order of magnitude using TSNC compared to RLNC, and an approximately fixed energy consumption no matter which coding scheme the devices used. Thus, leading to our conclusion that the fastest coding scheme in terms of processing speed consumes least energy per encoded or decoded bit. Another observation made from our measurements was that NC with SIMD enabled provided further speed-ups of 4x to 18x in processing speed.

Our work also proposed a scheme of overlapping generations based on the ideas of our feedback based TSNC scheme. Using practical SRLNC measurements from a real-life implementation, we estimated its performance to be potentially an order of magnitude faster than multiple non-overlapping generations of systematic RLNC, when trading off delay performance. These results might provide an incentive to test our scheme in a real-life implementation.

We are yet to see a widescale assimilation of NC, this could be due to 1) the computational complexity of NC in practical implementations, and 2) the required know-how and effort required to implement NC at the application layer, and 3) the large effort and long term process of deploying NC in lower network layers. We therefore proposed a radically different approach to utilize NC in network communication, by implementing it at the filesystem layer in end devices, i.e., as a filesystem or a shim between user space applications and regular filesystems. We demonstrated how NC could be implemented as a simple protocol, on the top of regular and even proprietary applications, to

add multipath and multi-source capabilities to legacy HTTP communication. The implementation was deployed in Linux using the FUSE library. From practical tests, we observed two to five fold speed-ups when downloading 10 MiB and 100 MiB files from multiple server mirrors simultaneously, compared to traditional downloading mechanisms. These ideas could potentially form the basis for novel ways to easily extend existing protocols, e.g., with NC, as the Internet infrastructure continues to become increasingly wireless and distributed. This ongoing paradigm shift should lead to even more coding opportunities in future networks, and it will certainly bring a vast amount of novel applications that could potentially benefit from NC.

This thesis advocated for the use of sparse NC schemes in particular, which appeared very promising as a means to permit NC to run in devices with limited computational capabilities, e.g., IoT devices, whereas RLNC may remain the first choice in powerful devices that demands very low delay performances. TSNC that we mainly addressed in this thesis, proved to be relatively simple and flexible to deploy as an intermediate between very sparse codes, i.e., LT and Raptor codes, and very dense codes, i.e., RLNC, while potentially allowing recoding at intermediate network nodes.

In future work, TSNC could beneficially be used in multicast and multi-source scenarios, e.g., in combination with our OG scheme from Paper C and/or our network coded filesystem shim from Paper D. Furthermore, our basic feedback scheme of TSNC could potentially be improved or changed from pre-defined feedbacks towards feedbacks on demand. We imagine that TSNC sources could include an expected DOF in its packets to allow receivers to emit feedbacks only in case their rank differs significantly from that expected by the source. This approach may be more feasible in multicast applications as it could reduce feedbacks significantly and ease the decision of who and when receivers should feedback. Hence making feedback based TSNC simpler and more likely to be used in future applications. Most probably in multicast/broadcast scenarios, e.g., on stadiums, where lots of people are gathered to access the same data stream simultaneously.

References

[1] B. Adamson, C. Bormann, M. Handley, and J. Macker, "Nack-oriented reliable multicast (norm) transport protocol," Internet Requests for Comments, RFC Editor, RFC 5740, November 2009.

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

[3] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," Internet Requests for Comments, RFC Editor, RFC 7540, May 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7540.txt

[4] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of Allerton Conference on Communication, Control, and Computing*, October 2003.

[5] S. Feizi, D. E. Lucani, C. W. Sørensen, A. Makhdoumi, and M. Medard, "Tunable sparse network coding for multicast networks," in *Network Coding (NetCod), 2014 International Symposium on*, June 2014, pp. 1–6.

[6] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *Proc. of the Int. Zurich Seminar on Comm.*, March 2012, pp. 107–110.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," Internet Requests for Comments, RFC Editor, RFC 2068, January 1997.

[8] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[9] P. Garrido, C. W. Sørensen, D. E. L. Roetter, and R. Aguero, "Performance and complexity of tunable sparse network coding with gradual growing tuning functions over wireless networks," *I E E E International Symposium Personal, Indoor and Mobile Radio Communications*, 2016.

[10] A. Heidarzadeh and A. H. Banihashemi, "Overlapped chunked network coding," *CoRR*, vol. abs/0908.3234, 2009.

[11] J. Heide, M. Pedersen, and F. Fitzek, "Decoding algorithms for random linear network codes," *Lecture Notes in Computer Science*, vol. 6827, pp. 129–137, 2011.

[12] N. J. Hernández Marcano, C. W. Sørensen, J. A. Cabrera G., S. Wunderlich, D. E. Lucani, and F. H. P. Fitzek, "On goodput and energy measurements of network coding schemes in the raspberry pi," *Electronics*, vol. 5, no. 4, p. 66, 2016. [Online]. Available: http://www.mdpi.com/2079-9292/5/4/66

[13] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, Oct 2006.

[14] A. L. Jones, I. Chatzigeorgiou, and A. Tassi, "Binary systematic network coding for progressive packet decoding," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 4499–4504.

[15] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271–280.

[16] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, Feb 2001.

[17] D. E. Lucani, M. Medard, and M. Stojanovic, "Random linear network coding for time-division duplexing: Field size considerations," in *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, Nov 2009, pp. 1–6.

[18] J. Luo, P. T. Eugster, and J. P. Hubaux, "Route driven gossip: probabilistic reliable multicast in ad hoc networks," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 3, March 2003, pp. 2229–2239 vol.3.

[19] D. J. C. MacKay, "Fountain codes," *IEE Proceedings - Communications*, vol. 152, no. 6, pp. 1062–1068, Dec 2005.

[20] P. Maymounkov and N. J. A. Harvey, "Methods for efficient network coding," in *Proc. of Allerton Conference on Communication, Control, and Computing*, September 2006, pp. 482–491.

[21] M. V. Pedersen, J. Heide, and F. Fitzek, "Kodo: An open and research oriented network coding library," *Lecture Notes in Computer Science*, vol. 6827, pp. 145–152, 2011.

[22] M. V. Pedersen, D. E. Lucani, F. H. P. Fitzek, C. W. Sørensen, and A. S. Badr, "Network coding designs suited for the real world: What works, what doesn't, what's promising," in *2013 IEEE Information Theory Workshop (ITW)*, Sept 2013, pp. 1–5.

[23] J. Postel and J. Reynolds, "File transfer protocol," Internet Requests for Comments, RFC Editor, STD 9, October 1985. [Online]. Available: http://www.rfc-editor.org/rfc/rfc959.txt

[24] S. Puducheri, J. Kliewer, and T. Fuja, "The design and performance of distributed lt codes," *Information Theory, IEEE Transactions on*, vol. 53, no. 10, pp. 3740–3754, Oct 2007.

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

[26] 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.

[27] M. Xiao, T. Aulin, and M. Medard, "Systematic binary deterministic rateless codes," in *2008 IEEE International Symposium on Information Theory*, July 2008, pp. 2066–2070.

[28] S. Yang and R. Yeung, "Coding for a network coded fountain," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, July 2011, pp. 2647–2651.

Papers

Paper A

A Practical View on Tunable Sparse Network Coding

Chres W. Sørensen, Arash S. Badr, Juan A. Cabrera, Daniel E. Lucani, Janus Heide, Frank H. P. Fitzek

The paper has been published in the
European Wireless (EW), 2015.

Abstract

Tunable sparse network coding (TSNC) constitutes a promising concept for trading off computational complexity and delay performance. This paper advocates for the use of judicious feedback as a key not only to make TSNC practical, but also to deliver a highly consistent and controlled delay performance to end devices. We propose and analyze a TSNC design that can be incorporated into both unicast and multicast data flows. An implementation of our approach is carried out in C++ and compared to random linear network coding (RLNC) and sparse versions of RLNC implemented in the fastest network coding library to date. Our measurements show that the processing speed of our TSNC mechanism can be increased by four-fold compared to an optimized RLNC implementation and with a minimal penalty on delay performance. Finally, we show that even a limited number of feedback packets can result in a radical improvement of the complexity-delay trade-off.

1 Introduction

Transmitting data reliably in multicast sessions over lossy networks can incur in a large overhead, since each data packet lost by even one receiver may trigger a retransmission of that packet via the multicast link or via direct unicast requests of lost packets to the server. This results in additional delays, signaling, and reduction of end-to-end throughput. Forward erasure correction mechanisms that code the original data packets are key to reduce these effects. In particular, rateless codes, e.g., Raptor codes [1], Random Linear Network Coding (RLNC), provide a flexible approach to seamlessly service networks with varying number of receivers and loss conditions. The key to their potential is that the encoding is not planned to any specific configuration and could be adapted on the fly. These coded packets have a high probability of being useful to the different receivers, so it does not matter which packets are lost by individual receivers.

RLNC provides additional capabilities and potentially better throughput performance due to its ability to recode at intermediate nodes [2, 3]. However, its encoding and decoding operations are fairly complex when increasing the number of original data packets, n, as it depends on Gaussian elimination [4] for decoding. Other codes, such as LT codes [5] and Raptor codes [1], lower computational complexity by mixing less original packets in each coded packet at the cost of increased probability of receiving linearly dependent data packets. In contrast to RLNC, they rely on belief propagation decoding algorithms. Therefore, they need to rely on carefully designed density distributions to decide how many packets should be mixed together to averagely decode with low delay. This poses a good end-to-end solution, but because of the carefully designed coding structure, introduced by the density

distribution, recoding becomes challenging [6].
The potential to trade-off complexity and delay may be beneficial to adapt to characteristics of the end-devices and to different channel conditions experienced by the data flow. For example, it can allow us to lower energy consumption on battery power devices. Tunable Sparse Network Coding (TSNC) [7, 8] exploits the fact that coded packets are less likely to be dependent in the beginning of a transmission [9], i.e., when the receiver has less knowledge. As the receiver accumulates more linear combinations, it will be more likely to receive data that it already has (linearly dependent of that previously received). The authors of TSNC proposed to split a transmission of a group of packets into regions, such the code can be very sparse in the beginning, but the density is increased towards the end of the transmission.
This paper investigates the complexity and delay of a real-life implementation of TSNC leveraging feedback to target a specific delay performance penalty (or overhead target). This approach is particularly useful not only to manage the dynamics of the underlying data transmission and the code structure, but as a way to deliver very consistent performance to various applications, e.g., video streaming. The core idea of the approach is to target a given overhead while at the same time reducing the computational complexity at the time of decoding. The proposed approach is inherently adaptive and recomputes the density after each feedback packet based on the overhead target. Our work incorporates in the analysis and measurements the added complexity associated with estimating the density after a feedback event. Finally, we present measurement results from an implementation was carried out in C++ based on the Kodo network coding library [10]. Our results show that feedback events allows for overhead targets of a few data packets to be achieved with four fold the decoding speed while still using a standard Gaussian elimination decoder.

2 System Model

We consider a generation consisting of n original data packets, which are transmitted on a non-erasure unicast channel from source to sink. The source and sink initially agree on a desired number of coded packets that the source should transmit to provide the sink with n linearly independent packets. We refer to this number as the *budget*. The budget is the n linearly independent packets plus an overhead of coded packets that are lost either due to channel erasures or linear dependencies between coded packets. In our evaluations, only due to dependencies since the channel is lossless. We can therefore

describe the budget as:

$$B(n,d) = \sum_{i=0}^{n-1} \frac{1}{P(i,n,d)}, \tag{A.1}$$

where $P(i,n,d)$ is the probability of receiving an innovative coded packet after receiving i linearly independent packets out of n packets given a coding density, d. We use the upper bound for the innovation probability from Feizi et al [8] given as:

$$P(i,n,d) \geq 1 - (1-d)^{n-i}. \tag{A.2}$$

The budget can be based on time constraints of the source and/or the sink, but also processing capabilities of the sink. A budget close to n demands the source to generate very dense packets that are unlikely to be dependent, whereas a more relaxed budget results in the source generating sparser codes. In TSNC, the idea is to adjust the coding complexity over time, starting with sparse codes that becomes denser and denser towards the end of the generation. Finally, ending up with the sink recovering the generation after receiving the number of packets specified by the budget.

3 Proposed Approach

We propose to split the generation into a number of regions, k_t, that are defined by the number of linearly independent packets the sink has received. I.e. the rank of its decoding matrix. The sink and source initially negotiate a desired budget and the regions of the generation that is to be transmitted. We propose that the sink transmit an acknowledgement of its rank when ending each region. This can be beneficial for the source to deduce the remaining budget. We define the regions with a simple scheme where the sink acknowledge its rank when it has accumulated the following ranks:

$$s(k) = n \cdot \frac{2^k - 1}{2^k}, \qquad k = 0, 1, 2, \ldots, k_t. \tag{A.3}$$

Optimizing the values of $s(k)$ shall be the goal of our future work. The current choice is simple and provides more feedback packets towards the end of the generation. This is key to allow for more accurate tracking of the budget. After negotiating the regions and budget, it is time for the source to start transmitting from the generation. But, it needs to know the coding density before it can start generating coded packets. We find this density using bisection to estimate a fixed density for the density region that satisfy

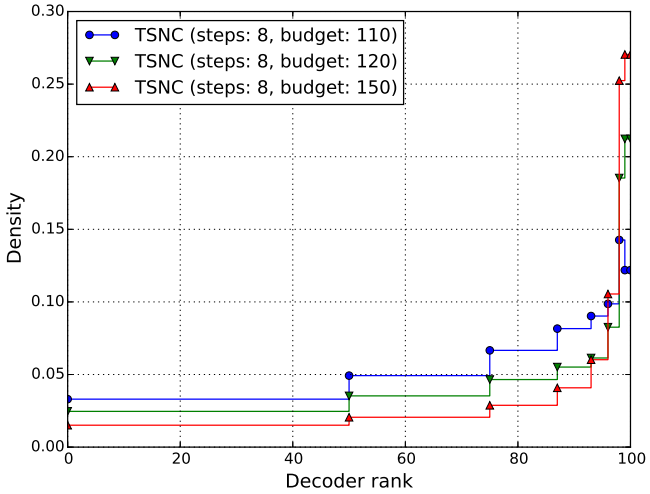


Fig. A.1: Average density of coded packets received when transmitting a generation of 100 original packets in eight density regions using TSNC and the field, $GF(2^8)$.

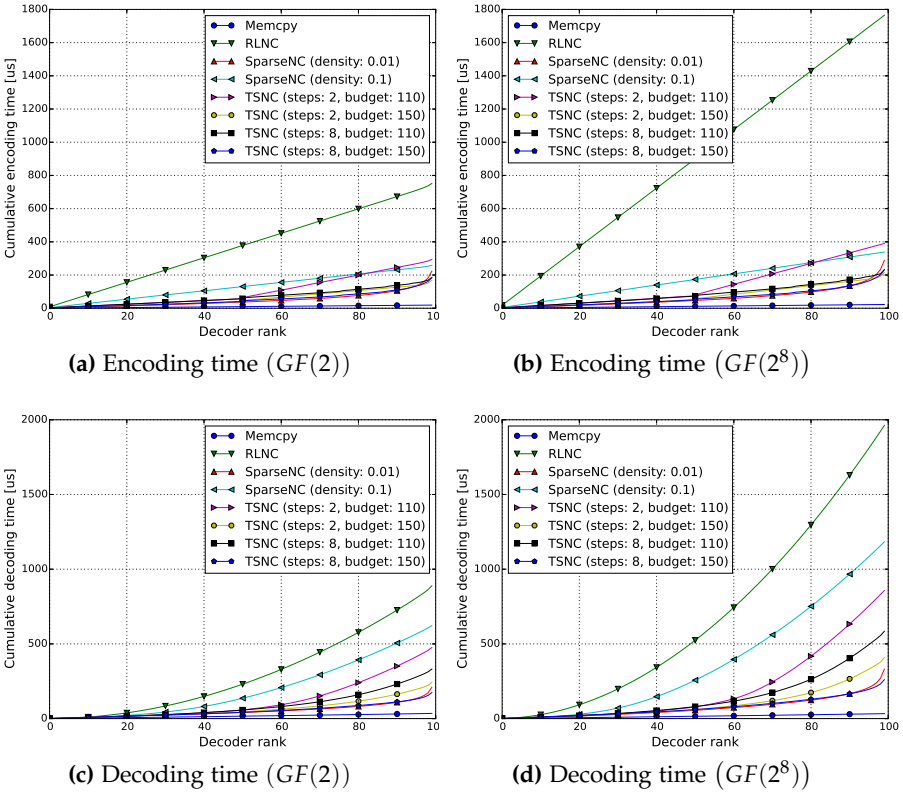
the budget for the k-th density region:

$$\begin{aligned} &B(s(k-1), s(k), n, d) \\ &= \sum_{i=s(k-1)}^{s(k)-1} \frac{1}{P(i,n,d)} = \sum_{i=s(k-1)}^{s(k)-1} \frac{1}{1-(1-d)^{n-i}} \\ &= \frac{B}{2^k}, \qquad \text{for } k = 1, \ldots, k_t - 1 \end{aligned} \tag{A.4}$$

where the budget is half of the remaining budget for all density region except of the last region which is assigned the entire remaining budget, i.e.,

$$\begin{aligned} B(s(k_t-1), s(k_t), n, d) &= \sum_{i=s(k_t-1)}^{s(k_t)-1} \frac{1}{P(i,n,d)} \\ &= \sum_{i=s(k_t-1)}^{s(k_t)-1} \frac{1}{1-(1-d)^{n-i}} = \frac{B}{2^{k_t-1}} \end{aligned} \tag{A.5}$$

Since the coding density may be low, we propose to send an uncoded data packet in case the encoder generates a coding vector that is all zero. This ensure that the source will always transmit data packets that are potentially innovative.



(a) Encoding time $(GF(2))$ **(b)** Encoding time $(GF(2^8))$

(c) Decoding time $(GF(2))$ **(d)** Decoding time $(GF(2^8))$

Fig. A.2: Cumulative encoding and decoding time to obtain a given rank.

4 Performance Comparison

We have implemented our proposed approach of Tunable Sparse Network Coding (TSNC) in the Kodo C++11 network coding library. This allow us to run a series of measurements to characterize the performance of a real life implementation of TSNC. We are particularly interested in measuring the goodput, i.e. linearly independent bytes per second, that can be processed by the encoder and decoder, but also to characterize how time is spend when transmitting a generation, i.e. will the estimation of the coding density impact the goodput. We measure this by transmitting generations of n original data packets. This is done one thousand times for all samples and averaged to reduce fluctuations. All measurements are conducted on a single computer that works both as source and sink in a lossless and delay free unicast network, to eliminate random channel behaviour such that focus can be narrowed down to encoding and decoding processes only. To put our results

into perspective, we compare the performance of TSNC against three other schemes:

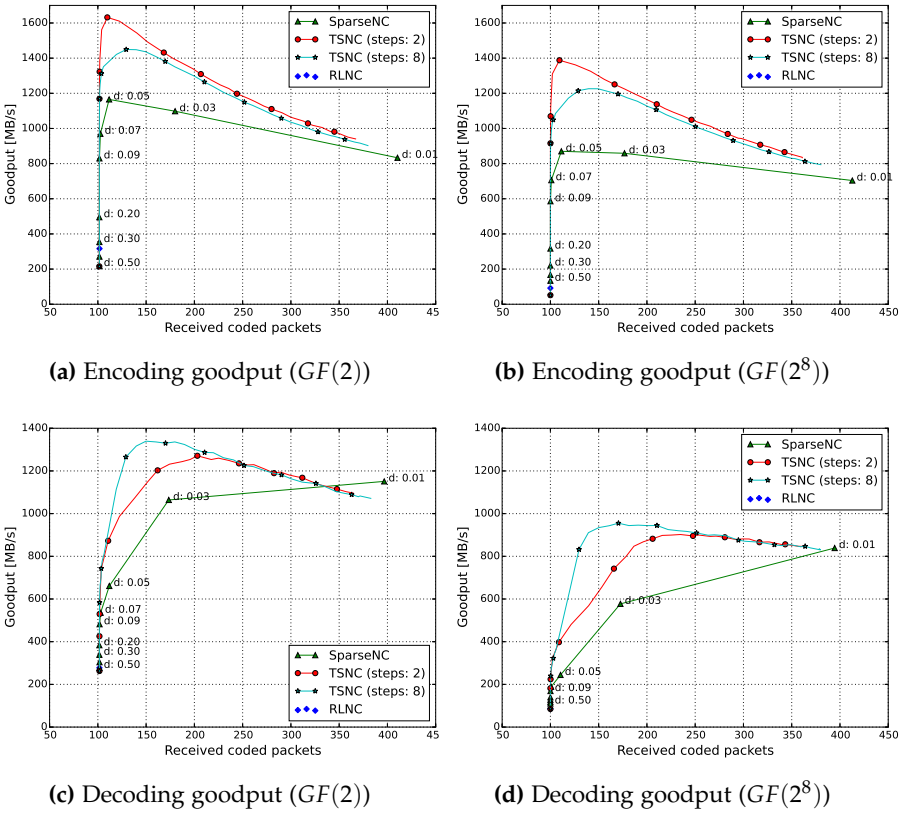
- *Transmitting the data uncoded and accounting only for memory copy (Memcpy)*. This scheme transmit all original data packets uncoded from the source to the sink. Since one computer runs both encoder and decoder, this is simply the same as just copying the n original data packets from one place in memory (on the source) to another place in memory (sink). This speed defines a lower bound of how fast we can possibly transmit a generation.

- *Random Linear Network Coding (RLNC)*. RLNC is the most dense code we can produce and thus very complex. It has a minimal delay of dependent coded packets. In fact, RLNC encoders produce 1.6 dependent coded packets on average for each generation if coding is performed in $GF(2)$. For higher fields, such as $GF(2^8)$, we can expect to generate close to no linearly dependent packets no matter the generation size.

- *Sparse Random Linear Network Coding (SparseNC)*. The objective of SparseNC is to trade-off complexity using a lower coding density, at the expense of a higher delay due to the increased probability of generating linearly dependent packets. SparseNC, keeps the coding density fixed throughout the transmission of an entire generation. Due to the sparse nature of the code, data packets are more likely to be linearly dependent as the sink accumulates more coded packets.

5 Measurements Results

In this section, we present the performance measurements of TSNC against the other three schemes. All measurements were performed on the same computer that was equipped with an Intel Core i7-3740QM processor running at 2.7 GHz.
Consider a source transmitting coded packets from a generation of $n = 100$ original data packets, each of 1500 bytes, to a sink in a lossless and delay free unicast network. This means that packets are lost only due to linear dependency and feedback information of the sinks rank is received instantly. The only times the source receives feedback from the sink is when the sink has accumulated enough linearly independent packets to finish a density region. This triggers a feedback which is used by the source to recalculate a new coding density. Figure A.1 illustrates the average coding density of coded packets received by the sink when receiving a generation of $n = 100$ original data packets using eight density steps and given three different budgets. It can be seen how the overall coding density is reduced for higher budgets



(a) Encoding goodput ($GF(2)$)

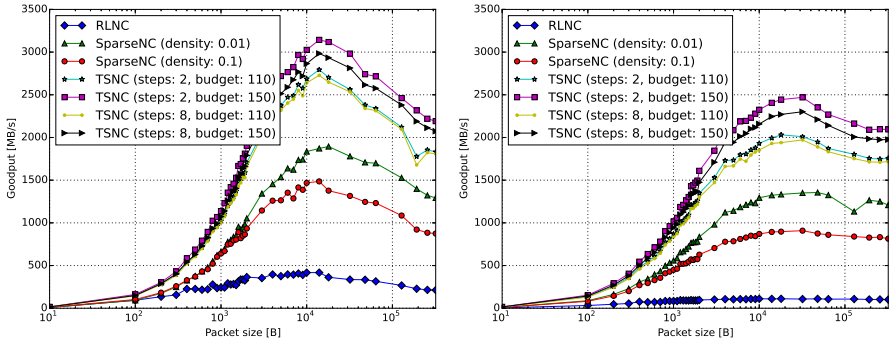
(b) Encoding goodput ($GF(2^8)$)

(c) Decoding goodput ($GF(2)$)

(d) Decoding goodput ($GF(2^8)$)

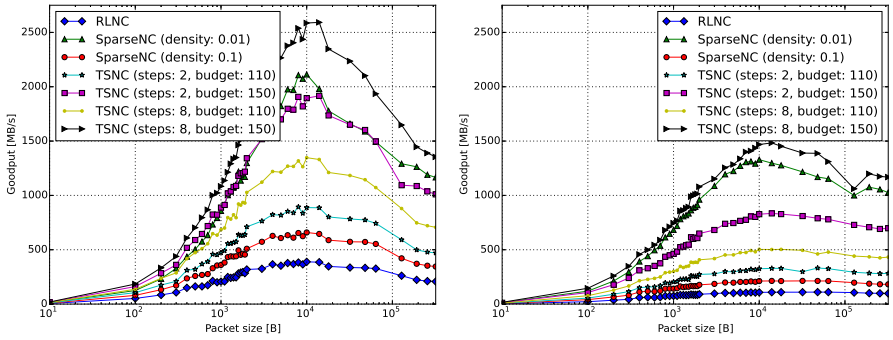
Fig. A.3: Encoding and decoding speed for a generation of 100 original data packets of each 1500 bytes.

and that the encoder grows the coding density in the end of the transmission to comply with the intended packet loss, due to linear dependency, in each density region. A question one might ask is how much the estimation of the coding density impact the performance of the source, and weather it is worth tuning the density instead of using a fixed density as SparseNC. Figure A.2 depicts the cumulative time of encoding (plots in the top) and decoding (plots in the bottom) as the sink accumulate more linearly independent packets. The plots show where time is spend when transmitting coded packets from a generation of $n = 100$ original data packets using the field $GF(2)$ (plots on the left) and $GF(2^8)$ (plots on the right). Since the parameters of SparseNC and TSNC may vary, we run SparseNC with two different coding densities and TSNC with two different number of density regions and budgets. This should provide an intuition of how the schemes



(a) Encoding goodput ($GF(2)$)

(b) Encoding goodput ($GF(2^8)$)

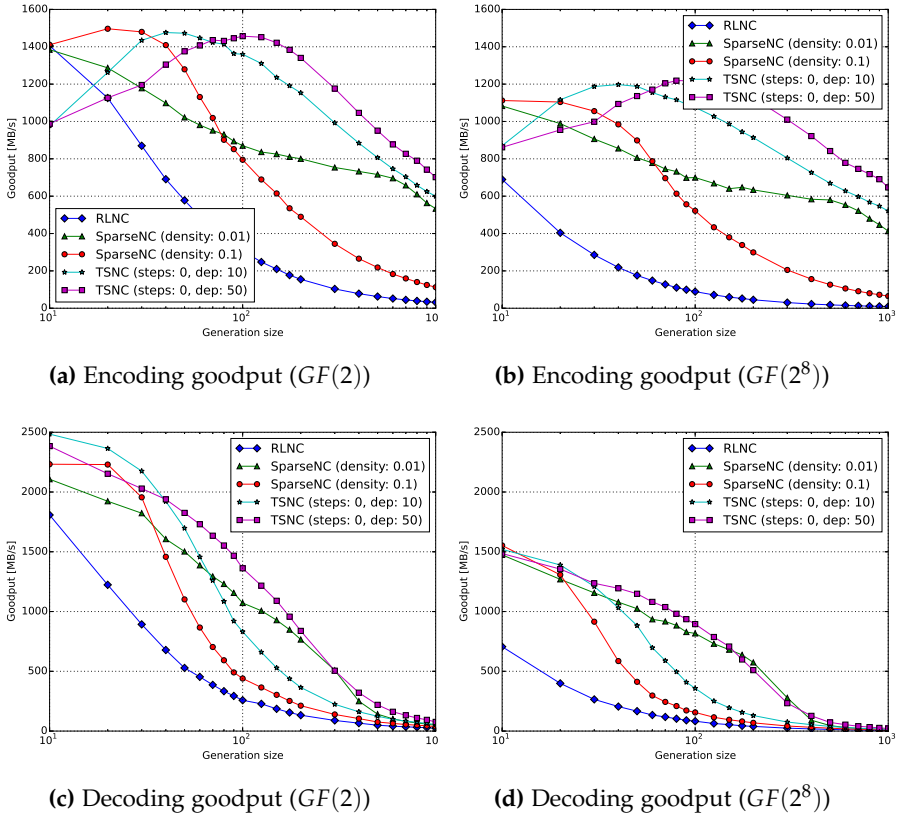


(c) Decoding goodput ($GF(2)$)

(d) Decoding goodput ($GF(2^8)$)

Fig. A.4: Encoding and decoding speed of generations of 100 original data packets of various size.

are affected by those parameters. The same values will be used as far as possible in the remaining simulations. From figure A.2, it can be seen that RLNC is slowest since it generates the most complex codes. Memcpy on the other hand is fastest since it only copy the original data packets in memory from the source to the sink. SparseNC and TSNC perform somewhere in between RLNC and Memcpy. It seem that sparser codes tends to run faster, both on the encoder and decoder, to a certain coding density at which the impact of dependent packets become too significant. This can be seen on the encoding time of SparseNC, with $d = 0.1$, which increase almost linearly while SparseNC, with $d = 0.01$, increase much faster towards the end of the transmission. This problem is attenuated in TSNC due to the coding density being increased for each density region. The estimation of the coding density can barely be seen in the plots although very small steps are present in the



(a) Encoding goodput ($GF(2)$)

(b) Encoding goodput ($GF(2^8)$)

(c) Decoding goodput ($GF(2)$)

(d) Decoding goodput ($GF(2^8)$)

Fig. A.5: Encoding and decoding speed of various sized generations with symbol of size 1500 bytes.

curves, of the encoding time, in the beginning of new density regions. Figure A.3 depicts how changing the coding complexity impacts the goodput and the number of coded packets required to decode a generation of $n = 100$ original data packets. Once again, the plots for $GF(2)$ are on the left side and the plots for $GF(2^8)$ are on the right side. RLNC form only one point near $n + 1.6$ received coded packets for $GF(2)$ and near n received coded packets for $GF(2^8)$. The goodput is low due to the high coding complexity. SparseNC and TSNC have the means to trade-off coding complexity at the expense of coded packets that are more likely to be linearly dependent and hence increasing the number of coded packets required to decode a generation. By measuring SparseNC with coding densities ranging from $d = 0.01$ to $d = 0.5$ for $GF(2)$ and to $d = 1$ for $GF(2^8)$, we get a curve that may provide an idea of which coding densities provide the best trade-off between received coded

packets and goodput. It can be seen that SparseNC with a density, $d = 0.01$, allows the decoder to process packets extremely fast, but it requires almost four hundred coded packets on average to decode a generation of $n = 100$ original data packets. As the density grows, it can be seen that the goodput decrease as the number of coded packets needed to decode a generation goes towards $n + 1.6$ for $GF(2)$ and n for $GF(2^8)$. We plot two curves for TSNC when varying the budget. One for TSNC with two density regions and one with eight density regions. The budget is approximately equal to the number of received coded packets. We see that RLNC is slowest as expected for decoding along with SparseNC when its coding density approaches the coding density of RLNC. For the encoder, SparseNC is actually slower than RLNC when it has the same density as RLNC. This is because the algorithm that generates coding coefficients is slower in SparseNC and TSNC for high densities. It is of no concern since high coding densities are rarely used in practice for SparseNC and TSNC (See figure A.1). The performance of both SparseNC and TSNC peak by allowing a small number of dependent packets in each generation. This is the best trade-off between coding density and linearly dependent packets. TSNC seem to outperform SparseNC due to the carefully selected coding density that allow the source to transmit very sparse codes in the beginning and yet dense coded packets towards the end of the transmission. Not surprisingly, we observe that the goodput decreases, both on the encoder and decoder, when the sink drop too many coded packets due to linearly dependencies.

The coded packets used so far have been of 1500 bytes, but what happens if the packets were either smaller or bigger. Figure A.4 provides plots of the average goodput of the schemes for various packet sizes. As in the previous plots, RLNC seem to be slowest due to the high coding complexity, and TSNC seem to be ahead of SparseNC because of its effort to tune the coding density. The measurements peaks just above a packet size of 10^4 bytes. This is likely due to the cache sizes of the CPU that had a L1, L2, and L3 cache of size 32KB, 256KB, and 6144KB, respectively.

A final thing one may like to change is the generation size. Figure A.5 depicts how the goodput behaves for a variety of generation sizes. In these plots, we introduce two new definitions. First, $steps = 0$ which tells the implementation of TSNC to use as many density regions possible when transmitting a generation. Second, dep describes how many dependent packets the implementation of TSNC should attempt to target. It is related to the budget as $budget = n + dep$. From the plots, it becomes clear that the encoder outperforms the speed of the decoder. This is because the encoders complexity grows very little when creating coded packets consisting of only a few original data packets. The decoder on the other hand has to eliminate all coefficients in the coding vector using an on-the-fly Gauss-Jordan decoding algorithm [4] no matter if the coded packet is innovative or not.

6 Conclusions

This paper presented and characterized a practical scheme based on the concept of tunable sparse network coding (TSNC). At the core of the scheme, lies the judicious use of feedback to tailor the density of the code as the transmission progresses in order to deliver a target delay performance. In fact, we show that the complexity-delay trade-off can be significantly improved by the use of limited number of feedback packets per generation.

We provide and implementation in C++ in the Kodo network coding library to compare the performance of our TSNC mechanism with highly optimized RLNC implementations. Our measurements show that a four-fold gain in decoding processing speed is achievable while maintaining a low added overhead of a few additional coded packets to guarantee decoding. We also show that higher speed-ups may be possible by improving our density adaptation algorithm. Moreover, our results are based on a Gaussian elimination decoder leaving the opportunity to further speed up decoding by leveraging decoding algorithms specifically designed for sparse matrices, as suggested in [8]. Our future work shall focus on designing various policies for managing feedback in multicast flows, designing and implementing efficient decoders that exploit the sparse nature of the code, and incorporating our scheme in real multicast protocols, e.g., NORM [11].

Acknowledgements

This work was financed in part by the Green Mobile Cloud project granted by the Danish Council for Independent Research (Grant No. DFF - 0602-01372B) and TuneSCode project granted by the Danish Council for Independent Research (Grant No. DFF – 1335-00125)

References

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

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

[3] R. Koetter and M. Medard, "An algebraic approach to network coding," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 5, pp. 782–795, Oct 2003.

[4] J. Heide, M. Pedersen, and F. Fitzek, "Decoding algorithms for random linear network codes," *Lecture Notes in Computer Science*, vol. 6827, pp. 129–137, 2011.

[5] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271–280.

[6] S. Puducheri, J. Kliewer, and T. Fuja, "The design and performance of distributed lt codes," *Information Theory, IEEE Transactions on*, vol. 53, no. 10, pp. 3740–3754, Oct 2007.

[7] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *Proc. of the Int. Zurich Seminar on Comm.*, March 2012, pp. 107–110.

[8] S. Feizi, D. E. Lucani, C. W. Sørensen, A. Makhdoumi, and M. Medard, "Tunable sparse network coding for multicast networks," in *Network Coding (NetCod), 2014 International Symposium on*, June 2014, pp. 1–6.

[9] C. Sørensen, D. Lucani, F. Fitzek, and M. Medard, "On-the-fly overlapping of sparse generations: A tunable sparse network coding perspective," *I E E E V T S Vehicular Technology Conference. Proceedings*, 2014.

[10] M. V. Pedersen, J. Heide, and F. Fitzek, "Kodo: An open and research oriented network coding library," *Lecture Notes in Computer Science*, vol. 6827, pp. 145–152, 2011.

[11] B. Adamson, C. Bormann, M. Handley, and J. Macker. (2004, November) Negative-acknowledgement (NACK)-Oriented Reliable Multicast (NORM) Protocol. [Online]. Available: http://cs.itd.nrl.navy.mil/work/norm

Paper B

Leaner and Meaner: Network Coding in SIMD Enabled Commercial Devices

Chres W. Sørensen, Achuthan Paramanathan, Juan A. Cabrera,
Morten V. Pedersen, Daniel E. Lucani, Frank H. P. Fitzek

The paper has been published in the
IEEE Wireless Communications and Networking Conference (WCNC), 2016.

Abstract

Although random linear network coding (RLNC) constitutes a highly efficient and distributed approach to enhance communication networks and distributed storage, it requires additional processing to be carried out in the network and in end devices. For mobile devices, this processing translates into energy use that may reduce the battery life of a device. This paper focuses not only on providing a comprehensive measurement study of the energy cost of RLNC in eight different computing platforms, but also explores novel approaches (e.g., tunable sparse network coding) and hardware optimizations for Single Instruction Multiple Data (SIMD) available in the latest generations of Intel and Advanced RISC Machines (ARM) processors. Our measurement results show that the former provides gains of two- to six-fold from the underlying algorithms over RLNC, while the latter provides gains for all schemes from 2x to as high as 20x. Finally, our results show that the latest generation of mobile processors reduce dramatically the energy per bit consumed for carrying out network coding operations compared to previous generations, thus making network coding a viable technology for the upcoming 5G communication systems, even without dedicated hardware.

1 Introduction

In the year 2020, it is expected that the number of Internet connected devices will increase to around 50 billion devices from the current 12 billion devices in 2015, according to a Cisco estimate [1]. A majority of those devices will connect wirelessly to the Internet, and their capabilities will be radically different, ranging from smartphones and laptops to small sensors. This dramatic increase will put a huge strain on the current network infrastructure. Thus, investigating and developing novel ways to deploy and operate communication networks has become critical in order to cope with the new requirements and offered network load.

Network coding [2], has proven to achieve network capacity in various scenarios by allowing intermediate nodes in the network to re-encode incoming packets (without decoding them) [3]. Random linear network coding (RLNC) provides a distributed way to achieve this performance by simply combining incoming coded packets uniformly at random in a finite field [4]. Instead of transmitting original data packets one by one, network coding encourages the source to generate and transmit linear combinations of the original data packets instead. Thus allowing the source to create an endless stream of coded packets, i.e. linear combinations of the original data packets, that are linearly independent from each other with high probability (innovative). This makes network coding well-suited for wireless communications since a sink only needs to receive n linearly independent coded packets to re-

cover n original data packets. It does not matter which specific coded packets are lost in the channel.

A main drawback of network coding is the additional processing cost introduced by encoding at the source (i.e., generating the linear combinations of the original data packets), re-encoding at intermediate nodes, and decoding the coded packets. In general, these tasks are computationally expensive, which results in increased energy footprint for processing and may result in a bottleneck for processing the data in specific devices. This is particularly problematic in battery powered devices, such as smartphones or sensors, if the gains from transmitting less packets using network coding does not outweigh the processing cost. Recent studies have focused on characterising the processing speed and energy consumption of network coding in commercial devices [5, 6]. However, these studies have focused primarily on RLNC or simple XOR-based network coding [7, 8]. Identifying and characterizing alternative mechanisms to speed up processing in commercial devices, which results in a lower energy footprint, is critical to allow future devices and their services to benefit from network coding's capabilities.

This paper will investigate the performance of network coding and its energy footprint on eight devices (five of them battery powered), using the Kodo C++11 network coding library [9], which is to the best of our knowledge currently the fastest network coding library. The study focuses on two key speed-up approaches. First, the use of sparse network codes as a means to reduce the number of overall computations that the system needs to perform compared to RLNC. Second, the use of specialized, yet widely available hardware support in modern Intel and Advanced RISC Machines (ARM) central processing units (CPU). This feature allows finite field operations to be processed faster, thus benefiting all network coding schemes.

To address the first speed-up approach, we focus on two sparse schemes, namely, sparse RLNC and tunable sparse network coding (TSNC), which can be applied as an alternative to RLNC to reduce the work load on both the encoder and decoder side [10–12] [1]. To fully harvest the reduced work load made possible by the use of sparse codes, it is important that the implementations of both the encoder and decoder are tailored to each scheme. Although we rely on a highly efficient, sparse-aware Gauss-Jordan implementation [13] for decoding, we propose various algorithms for encoding sparse data and choosing the fastest one based on the coding density. Future work will focus on implementing more efficient decoding algorithms, e.g., based on belief propagation, to further improve the decoding speed [14, 15].

To address the second approach, we study RLNC's performance on various commercial platforms with standard operations and with built-in, hardware-

[1] Due to the many devices and three coding schemes that will be analyzed throughout this paper, it is not possible to provide all plots in this paper. However, we will make them available at arXiv in an extended version of this paper.

optimized operations. The latter relies on hardware that supports vectorization, known as single instruction multiple data (SIMD) [16, 17]. The most recent versions for Intel and Advanced Micro Devices (AMD) processors is the Advanced Vector Extensions (AVX2) and for ARM processors it is called Neon. The vectorization capabilities are provided to developers through assembly libraries that have intrinsics for higher level programming languages, such as C++. The underlying idea behind AVX2 and Neon is essentially the same. This optimization approach contrasts with previous work on network coding speed up through the use of graphic cards [18, 19] in two ways. First, the use of graphic cards limits the usefulness of the implementation to a narrow subset of devices (not as generic). Second, graphic cards are typically asymmetrical in their data flow: higher speed to receive data than to return data, which generates a bottleneck.

The rest of the paper is organised as follows. Section 2 presents the testbed and measurement approach used to measure the energy consumption of mobile devices while they perform encoding and decoding procedures. Then, Section 3 present the three coding schemes that will be evaluated throughout this paper. Section 4 studies how algorithms can be tailored to each individual scheme based on the density of its coded packets as well as the basic idea of SIMD as a means to speed-up finite field operations. Finally, measurement results are presented in Section 5 and conclusions are summarized in Section 6.

2 Measurement Setup

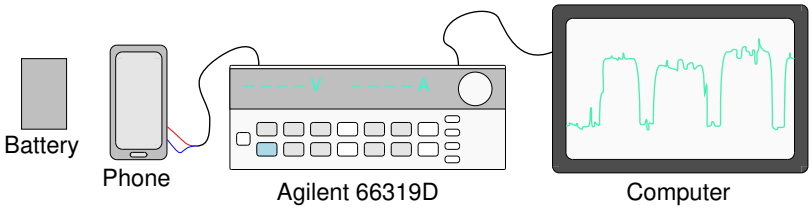


Fig. B.1: Testbed setup

This section will describe our setup to measure encoding and decoding speeds of the devices listed in Table B.1. The last four devices were powered by an Agilent 66319D, instead of their conventional power supply, to log their energy consumption while they ran network coding simulations. The setup is depicted in Figure B.1.

We wrote a script, that was deployed on each device, to sequentially run a series of simulations to estimate the encoding and decoding speed of the three coding schemes provided various configurations. The energy consump-

Table B.1: Measured devices

Alias	Device	CPU
N6	Nexus 6	Quad-core 2.7 GHz Krait 450
N9	Nexus 9	Dual-core 2.3 GHz Denver
i5	Intel NUC D54250WYK	Dual-core 2.6 GHz Intel core i5-4250U
i7	Dell latitude E6530	Quad-core 2.7 GHz Intel core i7-3740QM
Rasp	Raspberry PI 1 model B rev 2	Single-core 700 MHz ARM1176JZF-S
Rasp v2	Raspberry PI 2 model B V1.1	Quad-core 900MHz ARM Cortex-A7 CPU
S3	Samsung S3	Quad-core 1.4 GHz Cortex-A9
S5	Samsung S5	Quad-core 2.5 GHz Krait 400

tion looked roughly as illustrated on the computer monitor in Figure B.1. Because the coding speeds were stored in the measured device and the energy consumption was stored in a computer made it challenging to merge the data, since time drifted differently in the device and the computer. So instead of using timestamps to identify each simulation, we classified the electrical current samples into two groups based on the magnitude. Idle and simulation. The groups included samples in the transition phase between the two states. Those were eliminated by assigning the samples from the first and last few seconds of each group into a new transition group.

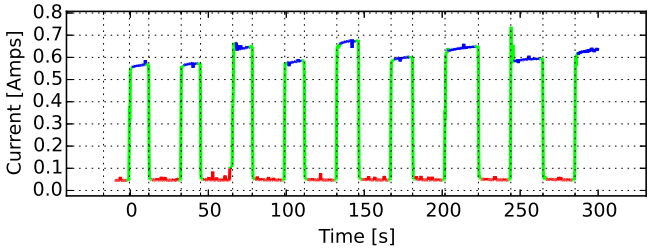


Fig. B.2: Current samples for Samsung S5. X-grid marks transitions between idle and simulation that are automatically detected by plotting scripts.

Coloring the groups confirmed that the samples had been classified correctly. This is shown in Figure B.2. The groups are colored as idle=red, transition=green, and simulation=blue.
Finally, each simulation was enumerated to map the coding speed measurements with the corresponding energy consumption. To extract the energy consumption of network coding, we subtracted the idle energy consumption from the energy consumption used during simulations.

3 Coding schemes

Three coding schemes will be presented in this section. RLNC, Sparse RLNC, and TSNC.

RLNC is known to be processing expensive, but it benefits from a minimal delay due to linearly dependency of coded packets. In fact, the decoder needs an average of only $n + 1.6$ coded packets using $GF(2)$ and n coded packets using $GF(2^8)$ to recover n original data packets. Each coded packet, CP, is generated using Equation B.1:

$$CP = \sum_{i=0}^{n-1} c_i P_i = \begin{bmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_{n-1} \end{bmatrix}, \tag{B.1}$$

where P_i is the i-th original data packet and c_i is the i-th coding coefficient that is assigned a randomly generated number drawn from a finite field. Considering a finite field, $GF(2)$, for this task means that each coefficient, c_i, is drawn uniformly from the set $\{0, 1\}$. This means that approximately half of the original data packets will be mixed into each coded packet. It significantly reduces the coding complexity compared to higher order finite fields for two reasons. 1) Coded packets are only a sum of approximately half of the original data packets, and 2) because multiplication is never performed when using $GF(2)$. This idea is exploited in Sparse RLNC, which works similar to RLNC, except that its coefficients, $\{c_i\}$, are generated with higher probability to be zero such that fewer original data packets are mixed into each coded packets. This makes the coded packets less complex to encode and decode, but it also makes them more likely to be linearly dependent. Thus, the receiver needs to collect more coded packets.

The last scheme is based on the idea that sparse coded packets are less likely to be innovative as the decoder accumulates more linearly independent packets. This is counteracted by gradually increasing the coding density over time. We define a budget, $B \geq n$, that describe how many coded packets a decoder should collect before it can recover n original data packets. Setting B a little larger than n allows the source to overall transmit coded packets that are less complex to decode, well knowing that it should transmit more coded packets to the sink. Our implementation of TSNC is based on [12]. The sink feedback its degree of freedom (DOF), i.e. number of linearly independent packets, when it has accumulated $s(k)$ DOF:

$$s(k) = n \cdot \frac{2^k - 1}{2^k}, \quad \text{for } k = 0, 1, 2, \ldots, k_t \tag{B.2}$$

where k is the k-th feedback that is transmitted during a generation. Due to the feedback packets, we consider the time between feedbacks as regions. Each region in our implementation will be assigned a part of the total budget such that a coding density can be estimated to fit the expected number of packets that should provide $s(k+1) - s(k)$ linear independent coded packets

to the sink. We assign the following budgets for each region:

$$B_{s(k-1),s(k)} = \frac{B}{2^k}, \quad \text{for } k = 1, \ldots, k_t - 1 \tag{B.3}$$

and for the very last density region:

$$B_{s(k_t-1),s(k_t)} = \frac{B}{2^{k_t-1}} \tag{B.4}$$

4 Implementation Optimizations of Coding Schemes

A coded packet is generated in two steps in the Kodo network coding library. 1) Decide the recipe of how to construct the coded packet. That is, generating a vector of coding coefficients that defines which original data packets are mixed into the coded packet.
2) Mix the original data packets together provided the vector of coding coefficients.

4.1 Generating coding coefficients

Which algorithm is fastest to generate vectors of coding coefficients depends on the coding density. This is illustrated by the speed of three different implementations in Figure B.3, that shows the time it takes to generate a vector of $n = 128$ coding coefficients in $GF(2)$. The density specifies the probability of each coefficient to be nonzero.

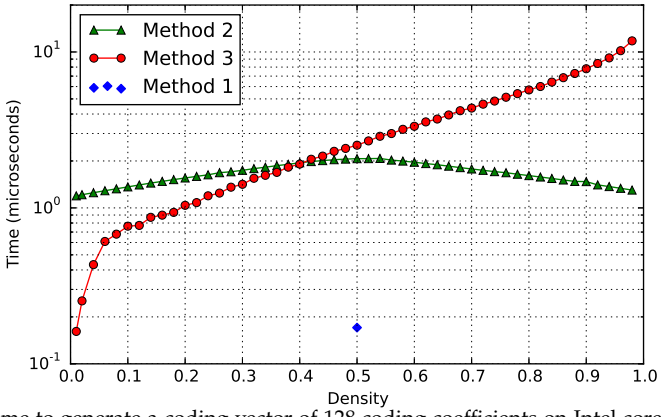


Fig. B.3: Time to generate a coding vector of 128 coding coefficients on Intel core i7 in $GF(2)$

Method 1: Works only for RLNC. It considers the coding vector as a block of memory and splits it into word sized elements that are assigned a uniform

random number one by one. This is very efficient because the word size is typically larger than the field size, allowing the algorithm to assign multiple coding coefficients simultaneously.

Method 2: Can be used to generate sparse codes for Sparse RLNC or TSNC. It loops over each coding coefficient and assigns a nonzero value with a probability specified by the density.

Method 3: Can be used to generate sparse codes for Sparse RLNC or TSNC. It generates a binomial random number, l, between 1 to n, and assigns nonzero uniform random values to coefficients randomly until l coefficients are nonzero.

Figure B.3 reveals that Method 1 is most efficient. This is due to its ability to assign multiple coefficients with each random number. This will however not work for sparse codes, since they demand various coding densities. It is therefore required to use one of the slower generators for that purpose. Since the sparse schemes are typically very sparse, it is expected that Method 3 is generally faster than Method 2. The measurements of sparse codes performed in this paper have therefore been generated using Method 3, which also has the advantage of guaranteeing that all zero coding vectors are never generated.

4.2 Mixing data packets

Provided a vector of randomly generated coding coefficients and the original data packets, the next task is to produce the coded packet according to Equation B.1. This involves a tedious process of additions and multiplications, that will benefit significantly from SIMD capabilities to perform a single operation on multiple data elements.
We will provide a simplified example of multiplying a coding coefficient with an original data packet with and without SIMD to illustrate the speed-up of SIMD. Consider a data packet of $M = 1600$ bytes, which is slightly larger than regular ethernet packets. Provided a finite field, $GF(2^8)$, each byte of the original data packet will be considered as an element that should be multiplied with the coding coefficient, c. The code snippet in Algorithm 1 illustrates a non SIMD approach to do this operation. $g(x)$ is an irreducible polynomial.

for $i \leftarrow 0$ **to** M **do**
| $cP[i] = c \cdot P_0[i] \mod g(x)$
end

Algorithm 1: Finite field multiplication

The code within the loop has to be executed $M = 1600$ times in this example, which is usually implemented as table lookups due to the irreducible polynomial. Each iteration takes time, and this is only a single multiplication among many additions and multiplications to produce the final coded packet. It is therefore crucial to use SIMD to optimize this type of repetitive task. SIMD allows a single operation to be performed on multiple data elements simultaneously in special registers. These registers takes blocks of 128 bits and 256 bits in Neon and AVX2 respectively, and allows arithmetics on elements of 8, 16, or 32 bits to be performed simultaneously on the registers full length. For Neon with 128 bits, this means that it can perform 16 multiplications simultaneously when each element is one byte. This idea is illustrated in Figure B.4. Since SIMD registers are 128 bits in Neon, we can produce a vector of the coding coefficient, c, repeated 16 times that is multiplied onto 16 elements of the packet at a time. This region is marked by a dashed rectangle. The example does not perform the operations over the irreducible polynomial, but rather presented the intuition of how SIMD works and why it is fast. [16] present how vectors can be generated and stored to perform finite field computations using SIMD. The decoder can apply this in the Gauss-Jordan elimination where subtraction is defined as bitwise XOR operations in finite fields and division is defined as multiplication with the multiplicative inverse.

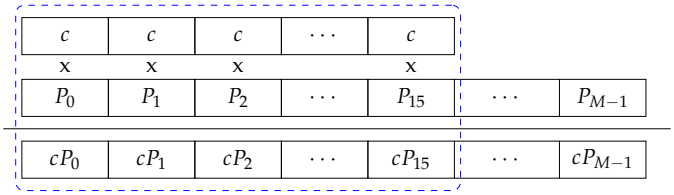


Fig. B.4: SIMD multiplication

5 Measurement results

This section will convey the performance measurements of encoding and decoding on eight devices listed in Table B.1. We considered three network coding schemes, RLNC, Sparse RLNC, and TSNC, using a wide range of configurations to account for most ways to setup each schemes. A testbed was used to measure the energy consumption of the last four devices, in Table B.1, while they encoded and decoded data. Those measurements were used to estimate the energy consumption per coded bit. The coding performance will be presented as goodput, i.e. linearly independent megabytes of data per second, that can be encoded or decoded per second. Based on the goodput and energy measurements, we derive the energy spend per encod-

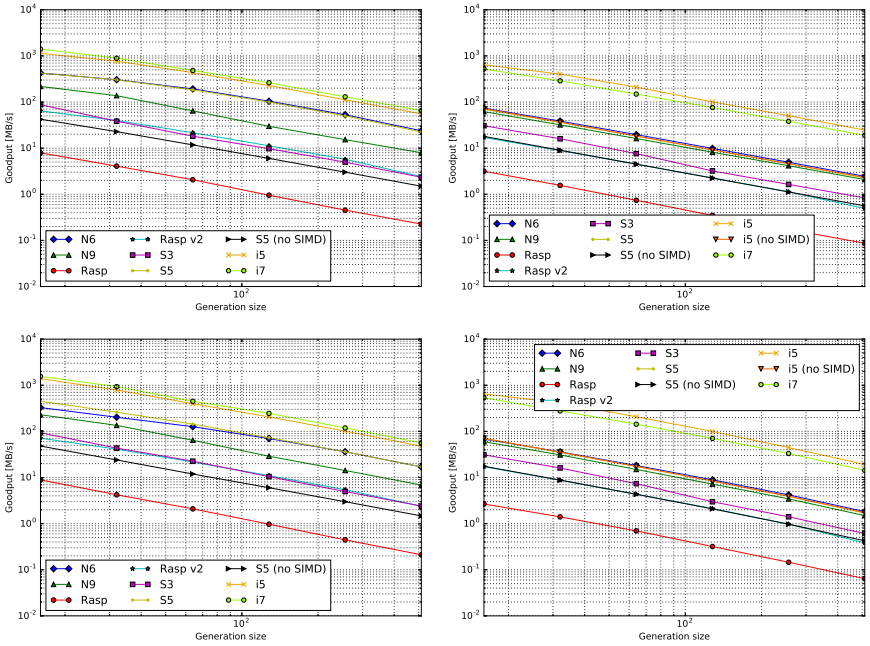


Fig. B.5: Goodput vs generation size of RLNC encoders (top) and decoders (bottom) using $GF(2)$ (left) and $GF(2^8)$ (right) with packet sizes of 1600 bytes.

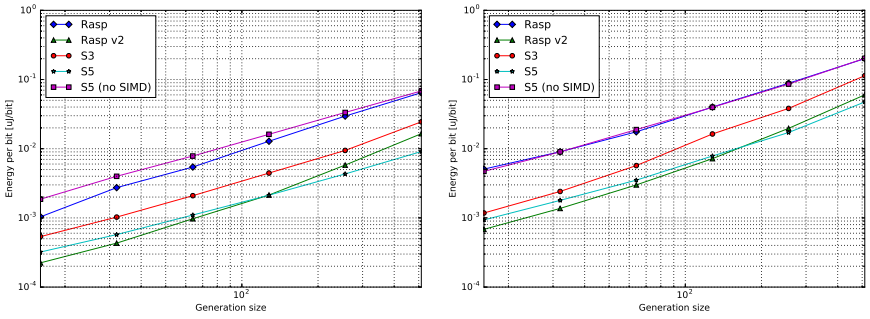


Fig. B.6: Energy per bit vs generation size of RLNC decoders using $GF(2)$ (left) and $GF(2^8)$ (right) with packet sizes of 1600 bytes.

ed/decoded bit by subtracting the energy consumed by a device in idle state from the energy consumed during simulations

Figure B.5 shows the goodput performance of encoding and decoding with RLNC in all devices, using both $GF(2)$ (left) and $GF(2^8)$ (right), with various generation sizes and a fixed packet size of 1600 bytes. We see that encoding and decoding becomes slower as the generation size increases, but

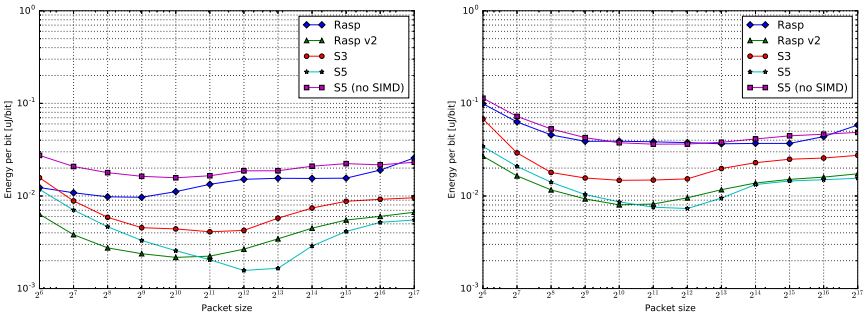


Fig. B.7: Energy per bit vs symbol size of RLNC decoding using $GF(2)$ (left) and $GF(2^8)$ (right) with a generation size of 128 data packets.

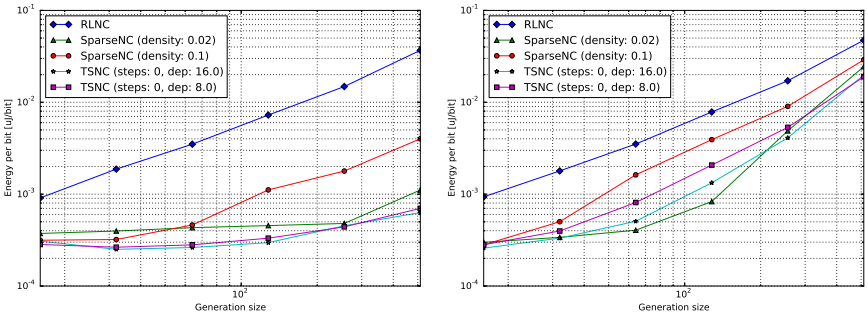


Fig. B.8: Samsung S5: Energy per bit vs generation size of encoding (left) and decoding (right) using $GF(2^8)$ with packet size of 1600 bytes.

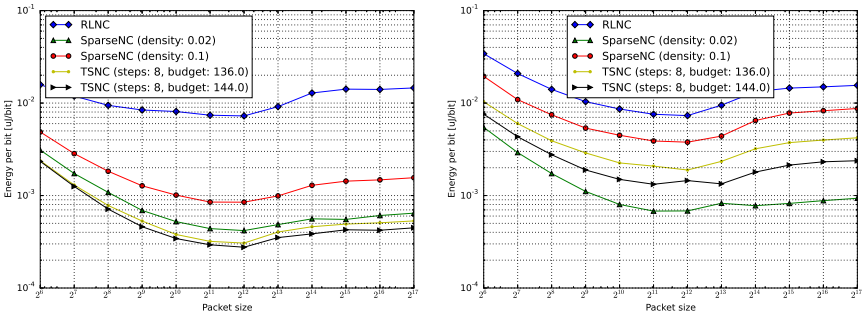


Fig. B.9: Samsung S5: Energy per bit vs symbol size of encoding (left) and decoding (right) using $GF(2^8)$ with generation size of 128 data packets.

also that the devices have the same trends although there are huge gaps in processing speeds among devices. That is due to different CPU clock frequencies, SIMD capabilities, system buses, cache sizes, and RAM speeds.

We notice that SIMD versus no SIMD provide gains between 4x and 18x, and that mobile devices that utilize SIMD compare equally with Intel core i5 without SIMD.

Based on the goodput measurements, Figure B.6 shows the energy per bit used by decoders using $GF(2)$ (left) and $GF(2^8)$ (right) for various generations sizes. The plots of encoding have been omitted since they reflected the same trends and almost same performance. Interestingly, it appears that Samsung S5 that had the highest goodput of the four devices is both the least and most energy consuming device depending on whether SIMD is enabled or not. This emphasizes the importance of utilizing the resources efficiently. The trends of all curves are mainly dictated by the goodput measurements since the energy consumption varies very little between simulations of different configurations. See Figure B.2

Figure B.7 shows the energy per bit again, but this time for a fixed generation size of 128 original data packets and with various packet sizes. As before, the plots shows measurements only for decoding using $GF(2)$ (left) and $GF(2^8)$ (right) due to the similar trends. The goodput plots have been left out, but should be reflected fairly well in the energy per bit plots. As in the latest plots, we see that Samsung S5 is both the least and most energy efficient device although it competes closely with Raspberry PI version 1 and version 2 to be the least and most energy efficient.

Figure B.8 considers the speeds of RLNC compared to the two sparser coding schemes in a Samsung S5. It is seen how the sparser codes benefits from reduced complexity although it comes with a higher probability that coded packets are linearly dependent. The packet size has been fixed to 1600 bytes, and we consider only $GF(2^8)$ since the same trends are reflected in $GF(2)$, although there are variations in the performance using the two fields. Again, dictated by the goodput, it is worth processing wise to transmit sparser packets that are less likely to be innovative. RLNC is by far slowest, and TSNC perform best due to its superior goodput achieved by tuning the density over time.

This is also the case for a variety of different packet sizes in Figure B.9 using 128 original data packets in $GF(2^8)$. Again, RLNC consumes far more energy, and based on the plots, it consumes 4x to 45x as much energy as the sparser codes for encoding, and 2.5x to 15x for decoding in $GF(2^8)$.

6 Conclusions

This paper characterized the processing speed performance and energy footprint of RLNC with different coding approaches on eight commercially available devices. Our comprehensive measurement campaign showed that (i) reducing the code's density can reduce the energy footprint by several fold

since sparser codes can be processed faster, and (ii) the use of generic hardware optimization (SIMD) already built-in in today's ARM and Intel processors can deliver order of magnitude processing speed-ups and energy per bit reductions of the same magnitude. In fact, we measured speed ups of up to 18x on the Samsung S5 with SIMD compared to not using SIMD. The latter is possible for any network code and is key to achieve the high performance demands of future services, while making energy use for encoding/decoding/re-encoding negligible compared to other processes.

Another key observation of our work is that high-end ARM processors, e.g., in the Samsung S5, with SIMD enabled are capable of delivering similar speeds to Intel core i5 processors without implementing SIMD. Furthermore, the energy footprint of these new processors is lower than previous versions, e.g., in the Samsung S3. In other words, there are already available mobile devices with the GB/s processing capabilities, with a negligible energy footprint (as low as 0.2 nJ/bit).

Acknowledgment

This work was partially financed by the green mobile cloud project (grant dff - 0602-01372b) and the tunescode project (grant dff – 1335-00125) granted by the danish council for independent research and the german research foundation (dfg) in the collaborative research center 912 ***h**ighly **a**daptive **e**nergy–efficient **c**omputing (haec)*.

References

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," Tech. Rep., 2011.

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

[3] R. Koetter and M. Medard, "An algebraic approach to network coding," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 5, pp. 782–795, Oct 2003.

[4] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, Oct 2006.

[5] A. Paramanathan, M. V. Pedersen, D. E. Lucani, F. H. P. Fitzek, and M. Katz, "Lean and mean: network coding for commercial devices," *Wireless Communications, IEEE*, vol. 20, no. 5, pp. 54–61, October 2013.

[6] A. Paramanathan, U. W. Rasmussen, M. Hundebøll, S. A. Rein, F. H. P. Fitzek, and G. Ertli, "Energy consumption model and measurement results for network coding-enabled ieee 802.11 meshed wireless networks," in *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2012 IEEE 17th International Workshop on*, Sept 2012, pp. 286–291.

[7] D. E. Lucani, M. Stojanovic, and M. Medard, "Random linear network coding for time division duplexing: Energy analysis," in *Communications, 2009. ICC '09. IEEE International Conference on*, June 2009, pp. 1–5.

[8] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "Xors in the air: Practical wireless network coding," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 3, pp. 497–510, June 2008.

[9] M. V. Pedersen, J. Heide, and F. Fitzek, "Kodo: An open and research oriented network coding library," *Lecture Notes in Computer Science*, vol. 6827, pp. 145–152, 2011.

[10] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *Proc. of the Int. Zurich Seminar on Comm.*, March 2012, pp. 107–110.

[11] S. Feizi, D. E. Lucani, C. W. Sørensen, A. Makhdoumi, and M. Medard, "Tunable sparse network coding for multicast networks," in *Network Coding (NetCod), 2014 International Symposium on*, June 2014, pp. 1–6.

[12] C. W. Sørensen, A. S. Badr, J. A. Cabrera, D. E. Lucani, J. Heide, and F. H. P. Fitzek, "A practical view on tunable sparse network coding," in *European Wireless 2015; 21th European Wireless Conference; Proceedings of*, May 2015, pp. 1–6.

[13] J. Heide, M. Pedersen, and F. Fitzek, "Decoding algorithms for random linear network codes," *Lecture Notes in Computer Science*, vol. 6827, pp. 129–137, 2011.

[14] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271–280.

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

[16] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast galois field arithmetic using intel simd instructions," ser. FAST'13, Berkeley, CA, USA, 2013, pp. 299–306.

[17] A. Limited, "Introducing neon," 2009.

[18] P. Vingelmann, P. Zanaty, F. H. P. Fitzek, and H. Charaf, "Implementation of random linear network coding on opengl-enabled graphics cards," in *Wireless Conference, 2009. EW 2009. European*, May 2009, pp. 118–123.

[19] P. Vingelmann and F. H. P. Fitzek, "Implementation of random linear network coding using nvidia's cuda toolkit," in *Networks for Grid Applications*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, A. Doulamis, J. Mambretti, I. Tomkos, and T. Varvarigou, Eds. Springer Berlin Heidelberg, 2010, vol. 25, pp. 131–138.

Paper C

On-the-fly Overlapping of Sparse Generations: A Tunable Sparse Network Coding Perspective

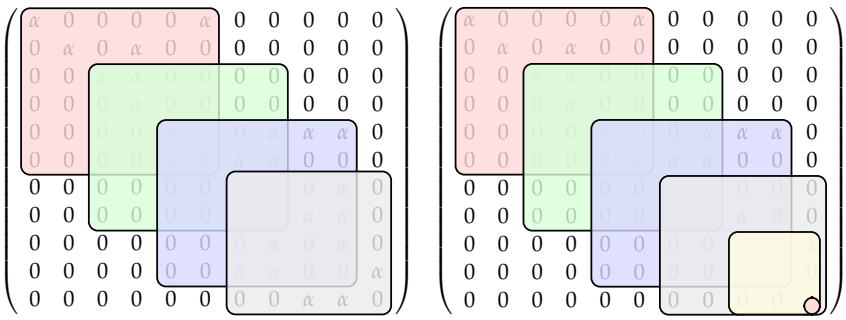
Chres W. Sørensen, Daniel E. Lucani,Frank H. P. Fitzek, Muriel Médard

The paper has been published in the
IEEE Vehicular Technology Conference (VTC Fall), 2014.

Abstract

Traditionally, the idea of overlapping generations in network coding research has focused on reducing the complexity of decoding large data files while maintaining the delay performance expected of a system that combines all data packets. However, the effort for encoding and decoding individual generations can still be quite high compared to other sparse coding approaches. This paper focuses on an inherently different approach that combines (i) sparsely coded generations configured on-the-fly based on (ii) controllable and infrequent feedback that allows the system to remove some original packets from the pool of packets to be mixed in the linear combinations. The latter is key to maintain a high impact of the coded packets received during the entire process while maintaining very sparsely coded generations. Interestingly, our proposed approach naturally bridges the idea of overlapping generations with that of tunable sparse network coding, thus providing the system with a seamless and adaptive strategy to balance complexity and delay performance. We analyze two families of strategies focused on these ideas. We also compare them to other standard approaches both in terms of delay performance and complexity as well as providing measurements in commercial devices to support our conclusions. Our results show that a judicious choice of the overlapping of the generations provides close-to-optimal delay performance, while reducing the decoding complexity by up to an order of magnitude with respect to other schemes.

1 Introduction



(a) Overlapping method **(b)** Overlapping method with variable n

Fig. C.1: Example of proposed overlapping generation methods. ($m = 11, n_{max} = 6, r = 4$)

The transmission of large amounts of data to multiple users in wireless networks requires mechanisms and protocols that are (i) resilient to packet losses, (ii) able to maintain a low overhead for transmissions, and (iii) adaptive to the network devices' heterogeneous capabilities and channel condi-

tions. Fountain codes, such as LT [1] and Raptor codes [2], pose a potential end-to-end solution to this problem. Since they exploit a belief propagation algorithm for decoding, receivers can implement a very resource efficient mechanism for decoding thus catering to a wide variety of devices. A key limitation of these codes is the fact that encoders need to follow a very strict density distribution to ensure decodability with low overhead (delay). Thus, LT and Raptor codes are useful only for end-to-end applications, which is inefficient in multi-hop scenarios.

Network coding provides an alternative solution by encouraging intermediate nodes in the network to operate on its incoming coded packets in order to generate new coded packets. The impact of recoding at intermediate nodes, i.e., coding in the network, allows to achieve the multicast capacity in lossless wireline networks and on lossy, multi-hop wireless networks. The latter comes in part from the ability to generate redundancy that is tailored to each wireless link, instead of generating redundancy end-to-end. Random linear network coding (RLNC) showed that recoding can be carried out in a distributed fashion by simply generating linear combinations of received packets using random coefficients drawn from a finite field [3]. In contrast, recoding capabilities in LT [1] and Raptor codes [2] at intermediate nodes has proven difficult to achieve without modifying the underlying code structure, e.g., [4].

A key limitation in RLNC lies in its decoding complexity, which is more resource expensive than belief propagation. In fact, given N packets of size K symbols in the given finite field, Gaussian elimination requires $O(N^3 + N^2K)$ operations to decode. Some approaches, such as systematic network coding [5] provide simple alternatives to reduce this complexity by sending uncoded packets at first, followed by RLNC packets later on. However, its applicability is typically limited to a few hops, as less uncoded packets will be received when traversing multiple, lossy links. From a practical perspective, complexity is reduced by splitting larger files into multiple disjoint generations of packets [6]. Thus, the system retains its recoding capabilities and maintains a complexity that is linear on the number of generations (although with a large constant), but at the cost of increased overhead. Generations can be transmitted sequentially or in a round-robin fashion [6] as well as by using a random schedule [7], using more or less feedback messages and smaller or larger storage, respectively.

The overhead introduced by splitting the file into smaller generations can be reduced by letting the chunks overlap [8, 9]. That way, an original packet may be contained within multiple generations. When a packet gets decoded within one generation, it may be back substituted into any other generation that contains it. This insight has spawned a variety of approaches from considering overlaps of generations with different sizes [10] to trade-off delay/overhead and complexity, to codes that use a sparse pre-coder before

creating generations, e.g., BATS codes [11]. Existing approaches have relied in the use of RLNC for coding within generations and an attempt to restrict the use of feedback in the transmission process.

This paper advocates that exploiting sparse coding within generations, instead of RLNC, and leveraging occasional feedback is instrumental to generating overlapping generations on-the-fly and providing a low complexity, low overhead solution. More generally, our approach allows us to trade-off the overhead in the use of the channel with decoding complexity to allow resource limited devices to exploit network coding. Our proposal is inspired in part by the results of [12] in Tunable Sparse Network Coding (TSNC) and its potential for recoding sparse codes. In fact, our proposal constitutes a specific implementation of TSNC, where the coding density is increased by dropping original packets that have been "seen" at the receiver as part of the pool of packets considered to generate coded packets (using the notation in [13]).

This paper proposes and analyzes families of on-the-fly, sparsely coded generations and compares it to various non-overlapping and overlapping generations approaches. We focus our evaluation on delay/overhead performance as well as complexity. For the latter, we consider measurements on commercial platforms to understand the processing time required by the different approaches. We show that specific configurations of feedback and sparsity in our approaches can provide a low overhead solution with several fold to an order of magnitude gain in processing time compared to all other approaches.

2 Model and Preliminaries

2.1 System Model

We consider the case of a sender transmitting a large group of m data packets to a set of receivers over packet erasure channels. We order the packets with a given index with the lowest index assigned to the first packet in the file. The sender organizes the data packets in generations with a smaller subset of the packets. Coded packets are generated by using linear combinations of the packets in each generation choosing the coding coefficients in a sparse fashion, i.e., by choosing only a limited number of non-zero coefficients.

A receiver transmits feedback packets to signal to the sender which packets have been *seen* up to that point, using a similar notation to [13]. A *seen* packet constitutes a packet that has been included in a received linearly independent coded packet. Each linearly independent coded packet can provide a single and unique *seen* packet at the time of sending a feedback packet. Packets with lower index are prioritized to have a greater overlap across multiple

receivers. If r packets were not *seen* in the previous generation, we say that the overlap between two generations is r packets.

A signaling event, i.e., reception of feedback packets at the sender, is generated before the transmission of all packets in the generation has been completed. The signaling event triggers the creation of a new generation, which overlaps with the previous one. The overlap is given by those packets that were not *seen* by all receivers. At this point, the sender can eliminate from its queue all packets that were *seen* packets by all receivers. This provides us with a coefficient matrix of the structure illustrated in Figure C.1a. Finally, feedback is assumed to be lossless and delay free, for simplicity.

2.2 Proposed Approaches

The use of sparse coding for the overlapping generations causes the probability of receiving a packet that is linearly independent of previously received packets to decrease as more coded packets of the generation are received. The main idea of letting generations overlap is to increase the innovation probability of coded packets, i.e., the probability of coded packets to be linearly independent, such that decoding can be performed with less received coded packets. Thus, generating a signaling event more often, i.e., increasing the frequency of feedback, results in a higher overlap between generations and, more importantly, in a lower overhead overall. The latter is a consequence of maintaining a high probability of receiving linearly independent coded packets during the entire transmission.

We propose two methods for overlapping generations based on the above paradigm. First, a method that defines a fixed generation size of n packets and a target overlap size of r. The last generation size will be lower or equal to the others in general. This method is referred to as OG in the remaining. Figure C.1a shows an example of this method in terms of the senders coding coefficients per sent generation. The example has $m = 11$ packets in total, which are split into smaller equally sized overlapping generations of $n = 6$ packets plus a potentially last generation of n packets or less. The generation overlap is $r = 4$ packets. Finally, the last generation will be $n_{\text{last}} = 5$ packets.

Since the last generation will be responsible for the highest overhead, i.e., additional received coded packets, it may be beneficial to reduce the sizes of the last generations as illustrated in Figure C.1b. This approach will be referred to as decreasing overlapping generations (DOG), and differs from overlapping generations (OG) by letting the overlap size, r, decrease such that generations may shrink in the end.

2.3 Metrics

We will focus on two performance measures throughout this paper. First, the number of received packets required to decode all m data packets. This allows us to measure the overhead of the different schemes. Second, the decoding time required to decode the m data packets using commercial devices.

3 Analysis

This section presents an analysis for our proposed overlapping sparse generation schemes described in Section 2. We also provide a similar analysis for comparison schemes. We will start by defining an upper bound for the estimate of the probability of a coded packet to be innovative, i.e., linearly independent, to a receiver that has accumulated i linearly independent packets. This probability can be calculated for a generation size of n data packets and a density, d as

$$P(i,n,d) = P_{innovative}(i,n,d) \geq 1-(1-d)^{n-i}. \tag{C.1}$$

This bound was used in [14].

Using Eq. (C.1), the expected number of packets needed to be received to increase a decoders rank by one can be calculated by $1/P(i,n,d)$. With that in mind, we can derive the expected number of packets needed to be received to decode a generation.

If we use a single generation (SG), the expected number of received coded packets is

$$E_{\mathrm{SG}}(m,d) = \sum_{i=0}^{m-1} \frac{1}{P(i,m,d)}. \tag{C.2}$$

In contrast, a non-overlapping generation scheme (NOG) consisting of k disjoint generations, $(k-1)$ equally sized and one generation of the same size or smaller. The expected received packets required to decode can therefore be found as

$$E_{\mathrm{NOG}}(m,n,d) = \\ (k-1)\sum_{i=0}^{n-1} \frac{1}{P(i,n,d)} + \sum_{i=1}^{n_{last}-1} \frac{1}{P(i,n_{last},d)}. \tag{C.3}$$

The number of generations is given by $k = ceil(m/n)$. The last generation size is found to be $n_{last} = m-((k-1)n)$.

For our proposed OG scheme, if we consider Figure C.1a, we see that only the first $n-r$ packets of each generation, except the last one, are transmitted.

The last generation should however be transmitted as a normal generation. This means that the expected number of received coded packets is given by

$$E_{\mathrm{OG}}(m,n,r,d) = \\ (k-1)\sum_{i=0}^{(n-1)-r}\frac{1}{P(i,n,d)} + \sum_{i=0}^{n_{last}-1}\frac{1}{P(i,n_{last},d)}. \tag{C.4}$$

The number of generations can be calculated as $k = 1 + ceil((m/n)(n-r))$, and the last generation will have the size $n_{last} = m - (k-1)(n-r)$.

Four our proposed DOG scheme, there is a decreasing overlap size and the reduction in generation sizes, which means that

$$E_{\mathrm{DOG}}(m,n,r,d) = \sum_{j=0}^{k-1}\left(\sum_{i=0}^{(n_j-1)-r_j}\frac{1}{P(i,n,d)}\right). \tag{C.5}$$

The number of generations by $k = ceil(m/(n-r))$. For each generation $j = \{0,1,\ldots,k-1\}$, we find the j'th generation size $n_j = min(n, m-j(n-r))$, and the decreasing overlap $r_j = max(0, n_j - n + r)$.

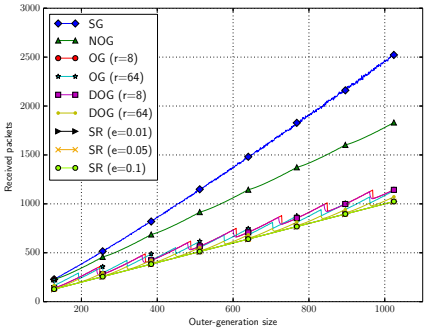
Finally, we consider a systematic approach with overlapping generations (SR). In each generation, the packets are first transmitted uncoded, and then finished using RLNC. The analysis is similar to the NOG scheme, where we have $(k-1)$ equally sized generations and one generation of same size or smaller. We complete one generation at a time, so we still sum the expected received packets required to decode in order to find a total amount of packets required to decode for m packets.

However, SR differs from the other methods since it does not consider a sparsely coded set of generations. Therefore, we are dependent on which packets are lost, and the packet erasure, e, has therefore been included into the expression

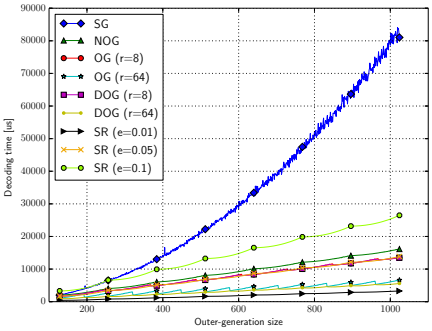
$$E_{\mathrm{SR}}(m,n,d,e) = \\ (k-1)\sum_{l=0}^{n}\left(\mathrm{Bi}(l,n,1-e)\left(l+\sum_{i=l}^{n-1}\frac{1}{P(i,n,d_{rlnc})}\right)\right) + \\ \sum_{l=0}^{n_{last}}\left(\mathrm{Bi}(l,n_{last},1-e)\left(l+\sum_{i=l}^{n_{last}-1}\frac{1}{P(i,n,d_{rlnc})}\right)\right). \tag{C.6}$$

where $Bi(l,n,p)$ represents a binomial distribution, where n is the generation size, p is the probability of successfully receiving coded packets, and $l = \{0,1,\ldots,n\}$ represents the number of uncoded packets received. We find the number of generations, $k = ceil(m/n)$, and the last generation size, $n_{last} = m - ((k-1)n)$.

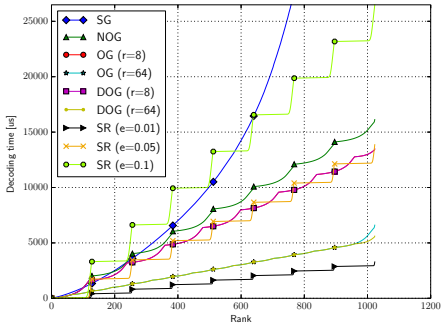
4 Performance Evaluation



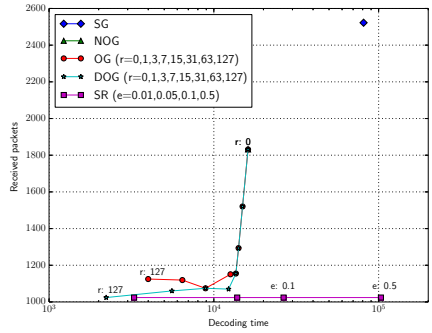
(a) Received packets required to decode as the outer-generation size is increased. (3-sparse, m={n..1024}, n=128, r={8,64}, e={0.01,0.05,0.1})



(b) Decoding time when the outer-generation size is changed. (3-sparse, m={n..1024}, n=128, r={8,64}, e={0.01,0.05,0.1})



(c) Time spend on decoding to obtain a given rank (3-sparse, m=1024, n=128, r={8,64}, e={0.01, 0.05, 0.1})



(d) Received packets as a function of decoding time (3-sparse, m=1024, n=128, r={0,1,3,7,15,31,63,127}, e={0.01, 0.05, 0.1, 0.5})

Fig. C.2: Results using $GF(2^8)/\{0\}$.

This section will be used to present the performance of our proposed methods, OG and DOG. The proposed methods will be compared to three other methods for transmitting a large group of packets: (1) transmitting all packets in a single generation using a 3-sparse density, named SG; (2) transmitting packets in smaller non-overlapping generations one generation at a time using 3-sparse, named NOG; (3) transmitting non-overlapping generations one at a time as in (2), but using systematic coding with RLNC to

complete each individual generation that experienced packet losses. We refer to this method as systematic RLNC (SR).

We have measured the average time spend decoding a generation until a given rank i is obtained. These measurements were performed on a decoder implemented in KODO [15]. Because we have the time spend to achieve a given rank, the measurements can simply be inserted in the equations of Section 3 to archive an estimate of the decoding time of the schemes given that they were implemented.

More specifically, we implemented a single-hop, single receiver setup and measured the time spend and the average number of symbols received by the decoder at each obtained rank. This was done for a single generation of size ranging from 1 to 1024 symbols, and a 3-sparse coding density such that $d = \min\left(0.5, \frac{3}{n}\right)$ for $GF(2)/\{0\}$ and $d = \min\left(1, \frac{3}{n}\right)$ for $GF(2^8)/\{0\}$. All measurements have been performed with packets of size 1500 bytes in $GF(2)$ and $GF(2^8)$, but the results will only be presented for $GF(2^8)$ since both fields show the same tendencies.

Given the measurement data, we can plot the packets required to decode an outer-generation of various sizes for each methods using the results from Section 3. This is illustrated in Figure C.2b, where the inner generation is $n = 128$ symbols is kept constant.

Figure C.2a shows that even with a density as low as 3-sparse, we obtain a performance that is far below SG and NOG, while performing only slightly worse than SR, which is optimal in terms of delay performance. Furthermore, DOG seem to perform slightly better than OG in terms of overhead, i.e., received coded packets.

Figure C.2b measures the decoding time of the same schemes. This time only measures the time invested in processing and, thus, is not affected by packet erasures on the communication channel. The packet erasures do however punish the SR method since an increased packet loss probability will cause more systematic packets to be lost and eventually replaced by RLNC packets. This is due to the fact that systematic packets require essentially no processing time, while RLNC packets are very dense and thus very time consuming to decode.

Figure C.2b shows also that SR performs better with low erasure probabilities, as expected, while OG and DOG perform better in case of increased erasures (> 5 %). We also see that a higher overlap is better in terms of decoding time. This may however change in a final implementation due to changes in the back-substitution and book-keeping mechanisms [16].

Figure C.2c considers the average decoding time it takes to obtain a given rank during transmission of an outer-generation of size $m = 1024$ packets. It is based on time measurements and generated using the equations presented in previous sections. Obtaining a rank is essentially the same as receiving an innovative packet, but does not mean that the packets can be decoded

yet. Again, we see the same tendencies as in the previous figures. The SR is very dependent on the erasure probability and will perform better in case of low erasures, but even with a relative small erasure probability it will be outperformed by OG and DOG.

Finally, Figure C.2d shows the explicit trade-off between received coded packets as a function of decoding (processing) time, considering the effect from the overlap size, r, and channel erasures on OG, DOG, and SR. The performance of SG is mediocre both in overall processing and delay performance, while the NOG method performance has similar performance to our proposed OG and DOG without overlap, $r = 0$. Increasing the overlap between generations, decreases the processing time on the overlapping methods due to less dependent packets. SR has the lowest probability of receiving dependent packets, but increasing the erasure even mildly will cause its decoding processing time to increase dramatically by more than an order of magnitude. Thus, DOG can provide close-to-optimal performance in delay (overhead) performance while providing a significantly smaller processing effort on the receivers. The feedback requirements for DOG and OG are mild and comparable in many cases to those of SR, i.e., a single feedback per generation used.

5 Conclusions

This paper advocates for an on-the-fly strategy for overlapping generations of data packets, while maintaining a sparse coding over the packets of each generation. More specifically, we propose two families of solutions that leverage a small amount of feedback to provide a controllable complexity-delay trade-off. Inherently, this article brings together the problems of overlapping generations and the tunable sparse network coding in a common setting.

Our comparison to alternative schemes were based on both delay/overhead performance and processing time on commercial devices. Our results showed that our proposed overlapping of sparse generation significantly decreases the number of received packets required to decode a large group of data packets. The level of overlap between generations has an important effect on performance, where a higher overlap maps into a better delay performance. We also showed that our proposed methods are very dependent on the last generation size, thus opening the door for future research in optimizing the generation sizes of the generations along the entire transmission process. Overall, we showed that our proposed methods can provide close-to-optimal delay performance, while reducing the processing effort by orders of magnitude in real systems.

Future work shall focus on more complex network settings, considering the effect of imperfect feedback, and considering the effect of recoding coded

packets at intermediate nodes.

Acknowledgements

This work was financed in part by the Green Mobile Cloud project granted by the Danish Council for Independent Research (Grant No. DFF - 0602-01372B) and TuneSCode project granted by the Danish Council for Independent Research (Grant No. DFF – 1335-00125)

References

[1] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271–280.

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

[3] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, Oct 2006.

[4] S. Puducheri, J. Kliewer, and T. Fuja, "The design and performance of distributed lt codes," *Information Theory, IEEE Transactions on*, vol. 53, no. 10, pp. 3740–3754, Oct 2007.

[5] D. Lucani, M. Medard, and M. Stojanovic, "Systematic network coding for time-division duplexing," in *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, June 2010, pp. 2403–2407.

[6] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of Allerton Conference on Communication, Control, and Computing*, October 2003.

[7] P. Maymounkov and N. J. A. Harvey, "Methods for efficient network coding," in *Proc. of Allerton Conference on Communication, Control, and Computing*, September 2006, pp. 482–491.

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

[9] A. Heidarzadeh and A. H. Banihashemi, "Overlapped chunked network coding," *CoRR*, vol. abs/0908.3234, 2009.

[10] Y. Li, W.-Y. Chan, and S. Blostein, "Network coding with unequal size overlapping generations," in *Network Coding (NetCod), 2012 International Symposium on*, June 2012, pp. 161–166.

[11] S. Yang and R. Yeung, "Coding for a network coded fountain," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, July 2011, pp. 2647–2651.

[12] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *Proc. of the Int. Zurich Seminar on Comm.*, March 2012, pp. 107–110.

[13] J. Sundararajan, D. Shah, and M. Medard, "Arq for network coding," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, July 2008, pp. 1651–1655.

[14] S. Feizi, D. E. Lucani, C. W. Sørensen, A. Makhdoumi, and M. Medard, "Tunable sparse network coding for multicast networks," in *Network Coding (NetCod), 2014 International Symposium on*, June 2014, pp. 1–6.

[15] M. V. Pedersen, J. Heide, and F. Fitzek, "Kodo: An open and research oriented network coding library," *Lecture Notes in Computer Science*, vol. 6827, pp. 145–152, 2011.

[16] J. Heide, M. Pedersen, and F. Fitzek, "Decoding algorithms for random linear network codes," *Lecture Notes in Computer Science*, vol. 6827, pp. 129–137, 2011.

Paper D

On Network Coded Filesystem Shim: Over-the-top Multipath Multi-Source Made Easy

Chres W. Sørensen, Daniel E. Lucani, Muriel Médard

The paper will be published in the
IEEE International Conference on Communications (ICC), 2017.

Abstract

Although network coding has shown the potential to revolutionize networking and storage, its deployment has faced a number of challenges. Usual proposals involve two approaches. First, deploying a new protocol (e.g., Multipath Coded TCP), or retrofitting another one (e.g., TCP/NC) to deliver benefits to any application in a computer. However, incorporating new protocols to the Internet is a challenging and slow process. Second, deploying coding at the application layer, which forces each application to implement network coding. This paper proposes an alternative approach through the use of a network coded filesystem shim (NCFSS), where coded data is generated at the filesystem level supporting any application and any network protocol. Our design allows multiple sources of a content to serve data without coordination to a receiver over multiple data paths. Another interesting feature of our approach is that it allows caches in the network to store only a fraction of a specific content in coded form, but sharing the same object identification, i.e., it simplifies the signaling and search of coded content. We describe the NCFSS' design and implementation using FUSE and carry out measurements using servers in six countries to demonstrate gains of two to five fold in download speed.

1 Introduction

Future communication networks will face tremendous challenges to answer to the increasing data traffic generated by end users, the novel requirements of 5G communications that go beyond higher data rates, e.g., low latency, mobility, ultra-reliability, and by the massive increase of network connected devices (expected to be between 28 and 500 billion in 2020 [1, 2]). These challenges require us to rethink the mechanisms and protocols that will enable emerging services and that can address some of the limitations of current protocols, e.g., Transmission Control Protocol (TCP) was not designed to manage mobility and is therefore challenged by handovers and suboptimal performance due to head-of-line blocking [3], handshake delay and channel erasures. From this perspective, NC provides an interesting technology to increase reliability and mobility support by using multiple paths and/or multiple data sources simultaneously as well as to improve and ease the operation of data caches closer to the end-devices, resulting in less latency to the end devices.

Research and experimentation in NC has focused on two key approaches to deliver services over the Internet: novel network coded protocols for data transport, e.g., [4, 5] and application-layer coding of the data that uses standard protocols for communication, e.g., [6] [1]. Despite these efforts and NC's

[1] We do not consider the research area on wireless mesh networks, e.g., [7, 8], because the scope is typically limited to the mesh itself and not on Internet-wide operation.

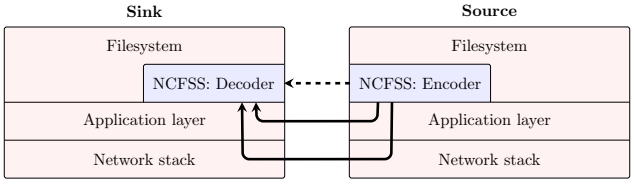


Fig. D.1: NCFSS in the filesystem. An application may copy data directly between filesystems within the same machine (e.g. using cp) or to a remote machine (e.g. using SCP or Wget).

inherent potential, the wide spread assimilation of NC is yet to happen at large scale. Part of this slow assimilation is related to the two technical approaches used so far. The former not only requires a large technical effort to design appropriate congestion and flow control, feedback management, protocol headers, and compatibility with the User Datagram Protocol (UDP)/TCP to ensure it can be deployed in the Internet, e.g., packets are not dropped by firewalls. It also requires a significant effort in standardization to be well-understood and accepted by the Internet community, which has dedicated significant efforts in the past to deliver stable and solid TCP improvements, e.g., increased reliability and throughput through Multipath TCP (MPTCP) [9, 10], which are backwards compatible with previous TCP versions, or even some recent protocols to replace TCP in specific applications, e.g., Quick UDP Internet Connections (QUIC) for web-browsing [11]. The advantage of such approach is that essentially any application would be able to use a network coded protocol. The second approach is limited by the fact that each application would need to be responsible for implementing NC, which limits the assimilation of the technology.

This paper presents a NCFSS to incorporate network coding capabilities in the filessytem (Figure D.1) to allow (i) any application to use coding (while being oblivious to it), (ii) nodes in the network to be backwards compatible by design, (iii) any end-device to draw data from multiple sources seamlessly, and (iv) rely on standardized protocols, such as TCP. Conceptually, our solution lies in between the previous two trends in NC practical research. Our approach can enable new capabilities, e.g., managing content from multiple servers/caches using multiple paths, that would be prohibitively complex otherwise.

Our contribution lies not only in proposing this new concept, but also in providing a proof-of-concept implementation in C++ using the FUSE [12] library. Our shim intercepts file I/O operations between any given application and a regular Linux filesystem and performs an on-the-fly alteration of the files that are read/written to/from an application and the filesystem. This paper illustrates the potential of this proof-of-concept to facilitate file transmissions over multiple flows in a simplified coded MPTCP like fashion. In contrast to MPTCP, our design enables each subflow to use any reliable

transport protocol, e.g.,TCP, QUIC, Stream Control Transmission Protocol (SCTP) [13]. In fact, the concept might be used to extend transmissions on legacy and even proprietary applications/protocols to use multiple flows. Using mirror servers in different countries, we carry out a real-life performance evaluation where a device retrieves data from multiple servers. Our measurements show that average gains of two to five fold are attained compared to state-of-the-art solutions, while maintaining a smaller standard deviation than downloads from a single server or uncoded downloads from multiple servers.

To the best of our knowledge, exposing a network protocol and coded data to user space applications as a means to extend networking capabilities has not been proposed before. Other filesystems or filesystem shims transparently read and write data from/to remote host(s) [14, 15], encrypt/decrypt data and/or distribute data over multiple hosts for redundancy, e.g. using NC [16, 17]. However, all of them expose data in its regular uncoded and readable form to the user space.

The paper is organized as follows. Section II describes key functionalities of FUSE and its use for developing our proof-of-concept implementation. Section III presents the concepts of multipath multi-source downloads that forms the basics for an underlying filesystem protocol used in our coded filesystem shim. Section IV presents an example usecase and details of how our coded filesystem concept may be used with FUSE in a regular directory structure to achieve multipath multi-source capabilities. Section V describes some caveats of the current implementation and discuss means to avoid or circumvent them in a refined implementation. Section VI describes the measurement setup and experimental results. Finally, Section VII provides the conclusions to our work.

2 The Network Coded Filesystem Shim

To understand NCFSS, let us introduce the basics of the Linux Virtual Filesystem (VFS) and FUSE.

2.1 Filesystems in Linux

A filesystem controls how data is stored and retrieved from a storage medium. The best filesystem implementation depends on the functionalities required as well as the usage, platform and storage medium of a system. This has led to the development of a large number of filesystems, e.g. the Extended Filesystem (EXT), B-tree file system (Btrfs), and Network File System (NFS) from Figure D.2.

In Linux, filesystems are encapsulated within the VFS whose purpose is to

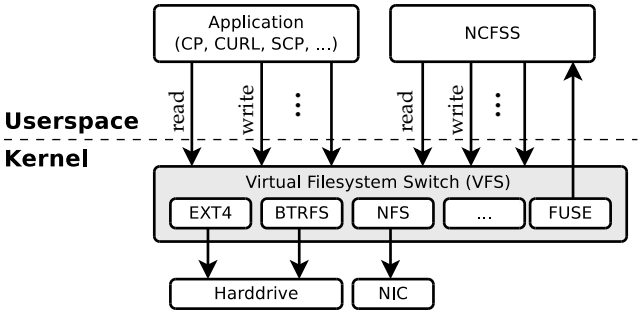


Fig. D.2: The NCFSS in userspace is compatible with the Linux VFS.

work as a uniform interface between userspace applications and filesystems. This allows applications to interact with any underlying filesystem using the same system calls regardless of the filesystem type. Another benefit of the uniform interface provided by the VFS is its ability to merge any number of filesystems into a single root filesystem.

2.2 FUSE to enable NCFSS

Filesystems generally reside in kernel space to achieve the best performance, but a filesystem may also reside in userspace. This can be achieved using the FUSE library. FUSE adds a module in the VFS that forwards system calls to a userspace application (in this case, NCFSS) that implements functions to handle the calls. A userspace filesystem is commonly used to develop filesystems that demands easy access to userspace libraries and/or Internet access, e.g., SSH Filesystem (SSHFS), GMAIL Filesystem (GmailFS).

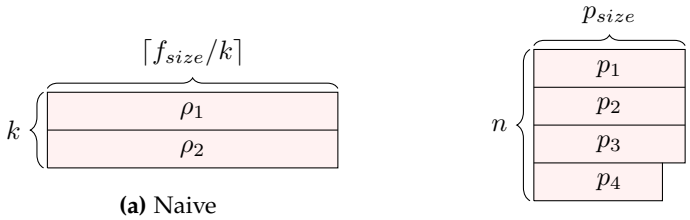
The majority of FUSE applications implement filesystem views. A filesystem view or virtual filesystem does not store files itself but rather provides access to files stored in another location. This could be in the same machine or another. Our work takes advantage of this approach to implement a shim that intercepts data between userspace applications and an existing filesystem in order to encode/decode the data going through the shim (NCFSS).

Although FUSE is written in C, it has bindings to C++, Python, and other programming languages and is supported in various Operating System (OS) (including, Linux, Android, OS X). We developed our proof-of-concept implementation in C++. FUSE is not supported in Windows, but there exists FUSE-like alternatives that enables porting the idea without implementing a filesystem in the Kernel.

3 Enabling Multipath Multi-Source Downloads

The main purpose of our shim is to facilitate multipath and multi-source download capabilities to existing filesystems. This section explains how a sink may split a file `f` to retrieve it using $k \geq 2$ sources simultaneously before describing how a simple NC protocol may be implemented and used in our shim. The file `f` is assumed to be fully available in all sources and we refer to its size in bytes as f_{size}.

3.1 Naive Multi-Source



(a) Naive

(b) Chunked

Fig. D.3: File abstraction for the naive and chunked downloading strategies for $k = 2$ sources.

This simplest way to retrieve `f` from k sources is to download one piece of the file from each source. Without knowledge of the transmission rates to/from the sources, this means that the sink must request roughly equally sized pieces from each source. The file can therefore be considered to consists of k pieces as illustrated in Figure D.3a. The size of the i-th piece is then

$$\rho_{\text{size}}(i) = \left\lfloor \frac{f_{\text{size}}}{k} \right\rfloor + \begin{cases} 1, & \text{if } i \leq f_{\text{size}} \bmod k \\ 0, & \text{otherwise} \end{cases} \quad [\text{bytes}]. \tag{D.1}$$

This means that the offset of the i-th piece, relative to the first byte of `f`, is given by

$$\rho_{\text{off}}(i) = (i-1) \left\lceil \frac{f_{\text{size}}}{k} \right\rceil \quad \text{for } i \in \{1, 2, \ldots, k\}. \tag{D.2}$$

This strategy can be easily applied on legacy protocols, such as the HTTP, but its performance is dictated by the slowest source. In fact, downloading from only one source may be faster than downloading from multiple sources in case just one of the sources are very slow. This problem is mitigated as the number of sources increases or by using the next strategy.

3.2 Chunked Multi-Source

Another strategy is to split `f` into even smaller pieces (also called chunks) that the sink may request one by one from the sources (Figure D.3b). This technique has proven to perform well in the BitTorrent network, and it is less sensitive to sources with heterogeneous transmission rates because a sink can retrieve more data pieces from faster sources.

The pieces can practically be of any size, but due to the signalling overhead from requesting pieces, it cannot be too small. In BitTorrent, the recommended piece size ranges from 32 KiB to 2 MiB depending on the file size.

The number of pieces n can be found given the file size and the desired piece sizes as

$$n = \left\lceil \frac{f_{\text{size}}}{p_{\text{size}}} \right\rceil . \tag{D.3}$$

Thus, the size of the i-th piece is

$$p_{\text{size}}(i) = \min\left\{p_{\text{size}}, f_{\text{size}} - (i-1)p_{\text{size}}\right\} \quad [\text{bytes}] \tag{D.4}$$

The offset of the i-th piece, relative to the start of `f`, is

$$p_{\text{off}}(i) = (i-1)\ p_{\text{size}} \quad \text{for } i \in \{1, 2, \ldots, n\}. \tag{D.5}$$

The smaller piece sizes enable a sink to fully utilize all sources while k or more pieces are missing. From that point on, the slowest source will dictate the remaining downloading time. In comparison with the naive strategy, this strategy demands a bigger effort on scheduling and requesting file pieces. Thus, resulting in additional bookkeeping and more network traffic from signalling. Another drawback is that it is not feasible to manually request the piece one by one. Instead, a dedicated application or script needs to be in control of requesting, retrieving and assembling the pieces in the sink.

3.3 Network Coded Multi-Source

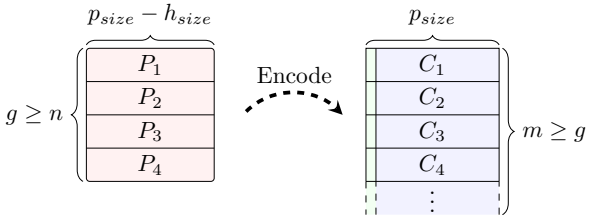


Fig. D.4: File abstraction for the network coded downloading strategies for $k = 2$ sources.

RLNC provides a radically different procedure to download `f` from multiple sources simultaneously. As discussed, the naive strategy was inefficient if even one source was slow while the chunked based strategy required a larger effort from sink to schedule, request, collect and re-assemble the data pieces received from the sources. The second approach also contributes with additional channel overhead from signalling. In contrast, using RLNC encoding and decoding at the sources and sink, respectively, enables each source to construct and transmit an endless stream of coded fragments of the original data pieces (Figure D.4). When received at the sink, any coded fragment is equally likely to be innovative irrespective of which source sent it. This means that a sink only needs to signal each source twice: once to request coded fragments and again when it has collected enough fragments to decode and thereby reconstruct the original data pieces. In this context, we define a coded fragment C_i as a linear combination of g original data pieces $P_j, j \in [1, 2, \ldots, g]$, as follows

$$C_i = \bigoplus_{j=1}^{n} v_{ij} \otimes P_j, \ \forall i \in [1, 2, \ldots), \tag{D.6}$$

where each v_{ij} is a random number uniformly drawn from a GF of size q. The sink can use Gauss-Jordan elimination on-the-fly to decode and thereby reconstruct all original data piece after it has collected g linearly independent coded fragments.

With RLNC, a sink needs to know how each coded fragment was constructed. There exists two common means to disseminate that knowledge. First, transmit the coding coefficients along with the codeword. Second, exchange the seed used to initialize the pseudo random number generator among each source/sink pair and include an id to each codeword such that the sink can reconstruct the coding coefficients to deduce how a coded fragment was constructed. NCFSS works using both methods, but this section will only describe the seed method due to an often smaller and simpler protocol header. The protocol is illustrated in Figure D.5. Each piece consists of an id and the codeword.

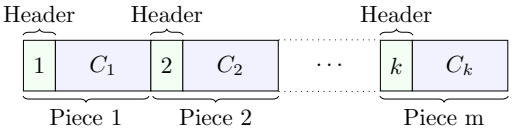


Fig. D.5: NCFSS protocol

Based on the protocol, it is possible to configure the encoder to construct coded fragments that fits within a data piece. Assuming that the id in the header is a fixed sized integer means that the size of each codeword can be calculated as

$$C_{\text{size}} = P_{\text{size}} = p_{\text{size}} - h_{\text{size}} \quad [\text{bytes}]. \tag{D.7}$$

Thus, the generation size is

$$g = \left\lceil \frac{f_{size}}{P_{\text{size}}} \right\rceil = \left\lceil \frac{f_{size}}{C_{\text{size}}} \right\rceil. \tag{D.8}$$

The benefits of RLNC are provided at the expense of additional computational complexity related to the coding and protocol overhead. The coding speed will not be addressed in this paper, but it should be aligned with the results obtained in [18] depending on the devices running NCFSS.

The overhead O is caused by 1) codeword id, 2) potentially additional bytes to zero-pad the last original data packet to fit into the generation, and 3) the possibility of receiving $r \geq 0$ linearly dependent packets at the sink. This results in an overhead that can be calculated using Equation D.9.

$$O = \frac{(g+r)p_{\text{size}} - f_{\text{size}}}{(g+r)p_{\text{size}}} = 1 - \frac{f_{\text{size}}}{(g+r)p_{\text{size}}}. \tag{D.9}$$

4 NCFSS by Example

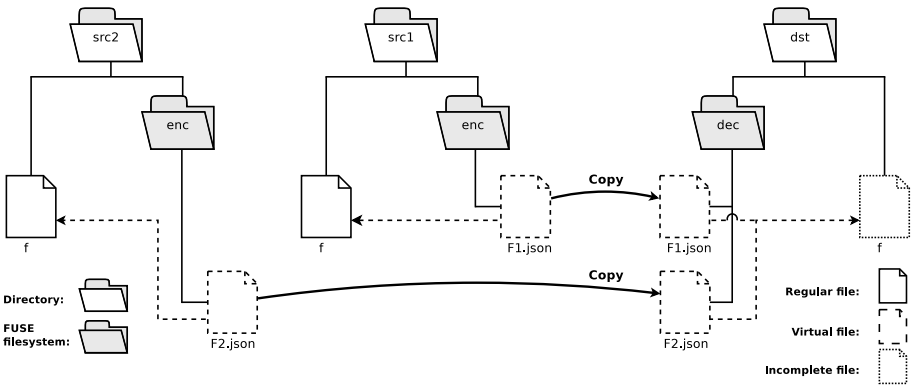


Fig. D.6: Filesystem operation to manage multiple data sources using `NCFSS` for encoding and decoding on the underlying filesystem

To understand the general operation of NCFSS, let us use an example with two data sources (`src1`, `src2`) and a single receiver (`dst`). The goal is for the receiver to recover a file `f` using the two sources simultaneously. Consider the directory structure in Figure D.6. The top directories `src1`, `src2` and `dst` each represent a regular directory. These directories could either be within a common filessytem structure (i.e., on the same host machine) or

part of different filesystem structures (i.e., on different host machines). In fact, this does not matter as long as the host machines use the Linux VFS. Adaptations for operation in other OS is possible, but will be beyond the scope of this work. We also assume that `src1/f` and `src2/f` are regular files that contain the exact same data and that `dst/f` is initially an empty or non-existing regular file. Thus, the overall objective of our work is to enable one or multiple applications to simultaneously copy the data in `src1/f` and `src2/f` to `dst/f`.

Using the previously described network coding protocol, we use FUSE to create NCFSS to transparently encode and decode regular data files on the fly. NCFSS is mounted on the directories `enc` and `dec`, which enables it to transparently observe and interpret all file I/O operations performed within those directories. Thus, enabling applications to create, remove, read, write and list files within `enc` and `dst` as in any other directories, but using NCFSS to 1) interpret file content when a file is created, 2) alter the way files appear within `enc` and `dec`, and 3) encode or decode data when it is read/written to/from files within the shim.

NCFSS use each file within `enc` to define how it should generate coded fragments from any regular data file whenever an application reads data from it. A file that is created within `enc` is interpreted by NCFSS upon creation and the virtual file needs to include a path to any regular data file that NCFSS should generate coded fragments from whenever the virtual file is being read. The content of the virtual file `F1.json` could be as illustrated in Code D.1. The virtual file is made in JSON format due to its human readability and ease of use, but any other desired formats could be implemented and used instead.

```
1  {
2      "piece_size": 32768,
3      "field": "binary8",
4      "seed": 237486,
5      "source": {
6          "path": "src1/f.txt",
7      }
8      "sink": {
9          "path": "dst/f.txt"
10     }
11 }
```

Code D.1: Content of `F1.json`

The current implementation of NCFSS immediately initialize an encoder that generates coded fragments from `src1/f` when an application reads from `enc/F1.json`. Another option could be to first initialize the encoder during the first read request. The data pieces being read from `enc/F1.json` by applications will adhere to the protocol defined in Figure D.5.

Due to the rateless nature of NC, it is not known in advance how many pieces the receiver needs to accumulate to reconstruct the regular data file. It all depends on the coding parameters specified in `F1.json`. It is therefore not possible to specify the exact number of data bytes that the sink needs to retrieve from `F1.json` to fully reconstruct `f`. Thus, the file size of `F1.json` is not fixed.

Fortunately, Linux does not require file sizes to be pre-defined. They can be presented to applications as 0 bytes although they return data when read. In fact, the Linux kernel uses a pseudo filesystem itself (Proc Filesystem (procfs)) to communicate with applications in userspace. The files in procfs typically appear to contain 0 bytes, but actually returns data. An example of such a file is `/proc/cpuinfo` that on-the-fly returns a detailed information of the system's CPU.

There may exist applications that demand file sizes to be non-zero or situations where it is desired to control exactly how many coded fragments a sink receives. NCFSS is able to compute a file size of any virtual file stored in `enc` and `dec` as

$$F_{\text{size}} = \left\lceil \frac{f_{\text{size}}}{p_{\text{size}} - h_{\text{size}}} \right\rceil p_{\text{size}} + r\, p_{\text{size}} = (g + r) p_{\text{size}} \quad [\text{bytes}], \tag{D.10}$$

where f_{size} is the size of the original file, h_{size} is the size of the header in each piece and r defines the number of redundant pieces. Thus, $r = 0$ means that the virtual file size should reflect exactly g pieces, $r > 0$ computes the file size to reflect $g + r$ pieces and finally, $r < 0$ to reflect $g - r$ pieces, i.e., in case an application only desires a subset of the code fragments.

Now that NCFSS is able to generate coded fragments of any file in a source, it is possible to use any application on the sink to retrieve the coded fragments. Keep in mind that the sink may retrieve a regular file by copying, e.g., `f` directly or coded through, e.g., `F1.json`. Coded data may be stored on the sinks harddrive in its coded state, but it may also be decoded on-the-fly using a RLNC capable application or using NCFSS mounted on `dec` as illustrated in Figure D.6.

In both cases, the decoder needs to be aware of how to decode the coded fragments. Similar to the `enc` directory, it is required to store `F1.json` also in `dec`. This allows NCFSS to initialize a decoder that transparently and on-the-fly decodes the coded fragments that are copied from `src1/F1.json` to `dst/F1.json`. When a piece of original data is fully decoded, it is automatically written to the regular file `dst/f`.

Because the pieces arrive coded, they are equally likely to be innovative no matter of the arrival order. This means that an additional flow may be constructed to facilitate multi-source downloads by creating `F2.json` as illustrated in Figure D.6. These files should contain the same coding parameters as `F1.json` but with a different seed and the file path for the source. NCFSS

will notice that it already has a virtual file `dst/F1.json` writing to `dst/f` and thereby know to share the same decoder.

On the source, NCFSS will always create one encoder per virtual file in `enc`, but regardless of the coding parameters, encoders may share the same source file. Thus, never storing more than one copy of each regular file in Random Access Memory (RAM). The ability to create multiple virtual files in `enc` and `dec` not only allows a sink to receive coded fragments from multiple sources, but it also enables both sources and the sink to transmit coded fragments over multiple network interfaces and using different network stacks.

5 Caveats and workarounds

There exists a few caveats with the implementation of NCFSS that may be addressed in future works. These caveats will be discussed in this section as well as potential solutions and/or workarounds.

5.1 Virtual/Configuration Files

NCFSS demands a virtual file (e.g. `F1.json`) to be created both in the `enc` and `dec`. That is necessary to initialize the encoder and decoder with similar coding parameters in the source and sink, respectively. This not only appears redundant, but it also causes problems to some applications that either terminates or rename the filename when copying to a file, e.g. `dst/F1.json`, that already exists.

NCFSS implements the ability to use two additional parameters in the virtual files, e.g. `F1.json`, to circumvent this challenges. 1) `"hidden": 1` hides the virtual file in `dec` from userspace applications, such that applications become unaware of its existence and therefore "overwrites" it as NCFSS expects, and 2) `"embed_config": 1` informs the source to pre-fix its virtual file in the data stream. The latter works in most applications because they deliver files reliable and in-order, but it is in practice not a safe workaround since it breaks the rateless property of NC. I.e. NC does not require pieces to arrive in order.

5.2 Terminating File Transfers

It may not be known in advance how many coded data fragments the sink needs to receive to fully decode a regular file. One challenge is therefore how a sink may terminate sources when a regular file has been fully decoded within NCFSS. To our knowledge, there exists no optimal or entirely clean way yet to do this, but we recommend that NCFSS indicate to the source(s) that the end of the file has been reached. However, it is not possible in the current implementation to signal the sources without two-way connections.

A less optimal solution that our current implementation relies on is to terminate the connections by throwing an error from the sink's shim to the application responsible for the file transfer. This causes the sink's application to terminate (often unexpectedly), but also the connection will be teared down and hence signaled to the source. The main issue with the latter is that there exist no correct error code for the purpose and it is also not similar how applications react on the different error codes.

5.3 Two-way Connections

Some of previously mentioned challenges may be resolved by extending NCFSS to support two-way communication. This may be accomplished by 1) implementing network capabilities into NCFSS, and/or 2) copying content not only from source to sink, but also from sink to source. Using an extended protocol, this would allow NCFSS to inject control messages in both directions.

6 Experimental Setup and Measurements

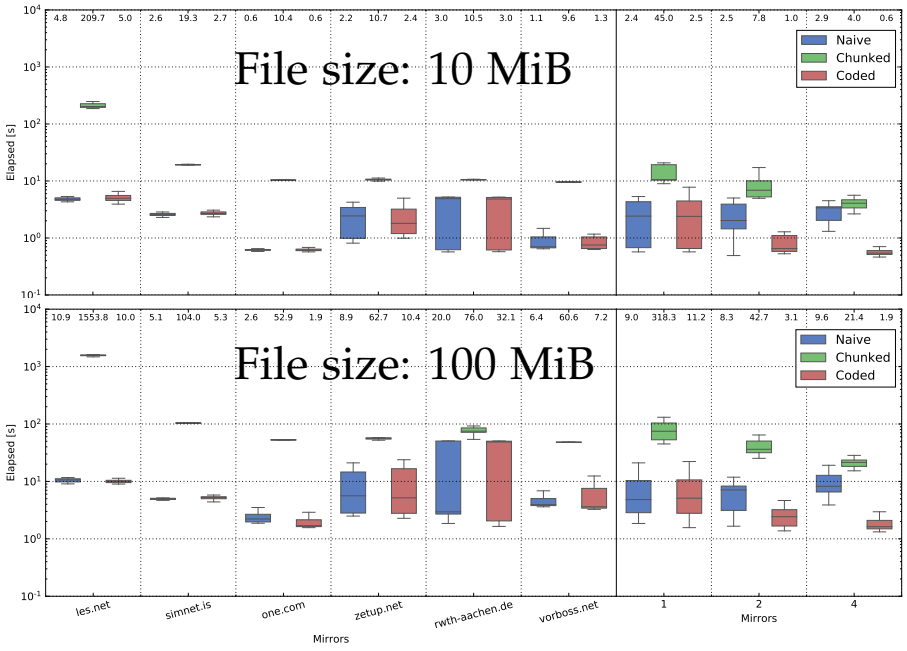


Fig. D.7: Comparing time to download 10 MiB and 100 MiB from a single HTTP mirror (left) and any one or multiple HTTP mirrors (right) using $p_{\text{size}} = 32$ KiB and $p_{\text{size}} = 64$ KiB for the two plot respectively. The numbers in the top of the plots show the mean of each box.

This section describes the measurement results from download tests over HTTP using our coded filesystem proposal. We have used six servers located in different countries in Europe and North America as HTTP servers with content of different sizes. For the results presented in this paper, we report the cases of 10 MiB and 100 MiB files. We carry out two types of measurements. First, measurements of the download time for the file content are performed from each server individually to understand its expected characteristic. Second, we carry out measurements where data is drawn from one, two, or four servers simultaneously using three downloading strategies. The set of measurements are performed over all potential combinations of the total number of servers, N, used for download. Thus, when downloading from k servers, we are considering $\binom{N}{k}$. For example, if $N = 6$ and $k = 2$, then we consider the 15 options available.

Figures D.7 present the download time for 10 MiB and 100 MiB files, when a sink downloads from each individual server in the left side plot and from any one, two or four mirrors in the right side plot. For the chunked and coded strategies, we used the piece sizes recommended for BitTorrent, which are 32 KiB for 10 MiB files and 64 KiB for 100 MiB files. We also used a 4 bytes header in each coded piece in the coded strategy as part of the embedded file protocol. This means that the coded strategy needs to retrieve at least 10.03 MiB and 100.06 MiB to recover the 10 MiB and 100 MiB files, respectively. Much of this overhead is due to the unfortunate zero-padding required to form the generation.

We observe that the naive download strategy is fastest when downloading from a single mirror, although closely followed by the coded strategy that has slightly more overhead due to the embedded file protocol. The speed of the chunked strategy is reduced considerably, as a consequence of piece requests, compared to the other strategies that only transmit a single request per source. When considering the file retrieval from multiple mirrors, it seems that the average download speed, using the naive strategy, is about 4% slower using two mirrors rather than one in the case of 10 MiB files, but 8% faster for 100 MiB files. Downloading from four mirrors on the other hand is about 16% slower than using only two mirrors. This is driven by the increased chance to include a slow mirror in the tests.

This problem of managing multiple mirrors (sources) is a good motivation to use the chunked or coded strategy instead as the performance of any individual server is less likely to drive the overall performance. Despite its low performance overall and, particularly, when using a single mirror, the chunked strategy is able to outperform itself using the fastest mirror by 19% for two mirrors and 60% for four mirrors. This follows a similar trend to our coded strategy, albeit 7 to 14 times slower than our coded strategy depending on the number of mirrors and data size.

On the other hand, the speed of our proposed coded strategy is 2.5 (2.7)

and 4.8 (5.0) times faster than the average speed of its fastest competing strategy for 10 MiB (100 MiB) files. Another benefit of coding is that it improves consistently when the sink retrieves coded pieces from more mirrors while its standard deviation decreases. This is supported by the box plot of Figures D.7, which shows fewer outliers and a concentration of measurements around the median.

Comparing the average download times of 10 MiB files against 100 MiB files show that it on takes approximately three to four times additional time to download 100 MiB compared to 10 MiB using the naive and coded strategies while the chunked strategy spends approximately six times longer for the download. This is driven by the cost for requesting content form each server using HTTP and the server response time that is not related to the network and download speed itself. This cost becomes more important in the overall system effect when downloading smaller files.

7 Conclusions

This paper presented a new alternative to deploy network coding in current and future networks that not only allows us to provide the multi-source and multipath capabilities that are crucial for ultra-reliable, highly mobile, and low latency data transport for emerging services. It also provides a natural approach to access and serve coded content with minimal or no coordination from the enabled applications. Our coded filesystem approach enables any application, communication protocol and standard filesystem or I/O operation to be supported, thus providing a backwards compatible approach to existing solutions and speedy deployment. Although our coded filesystem proposal relies on coded operations, it can also be useful without coding as a way to split and access different parts of the content without the need to include such operation at higher layers. However, using coding significantly reduces the cost of coordination to achieve the best performance.

Beyond advocating for a coded filesystem approach, this paper provided a proof-of-concept implementation using FUSE to deploy our solution. Our measurement results showed not only that coded filesystems can be deployed to support other higher layer protocols, e.g., HTTP, Secure Shell (SSH), but that the use of multiple sources can speed up access and download time of data by two to five fold with respect to the expected download time of multiple sources. Our approach also delivers a more consistent performance, i.e., smaller variance, specially when incorporating more sources.

In the future, the use of FUSE may not be required as the implementation could be part of the OS filesystem. This is also expected to reduce the cost of operations currently carried out in FUSE, as the copy from Kernel space to user space and back will be avoided. Our future work will aim to ex-

ploit the capabilities of coded filesystems using multiple caches with partial, coded data and demonstrating performance benefits on a number of other protocols. Our future work will also evaluate the performance of other code structures beyond RLNC [19] to reduce complexity at the end devices. Finally, our future work will go beyond the unidirectional flows considered in the experiments for this paper, e.g., for HTTP file downloads, to consider bidirectional data flows and protocols.

References

[1] D. Lund, C. MacGillivray, V. Turner, and M. Morales, "Worldwide and regional internet of things (iot) 2014 – 2020 forecast: A virtuous circle of proven value and demand," Tech. Rep., May 2014.

[2] J. Chamber, "Beyond the hype: Internet of things shows up strong at mobile world congress," Tech. Rep., 2014.

[3] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," in *IEEE Globecom*, Nov 2006, pp. 1–5.

[4] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard, "Network coded TCP (CTCP)," *CoRR*, vol. abs/1212.2291, 2012. [Online]. Available: http://arxiv.org/abs/1212.2291

[5] J. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets tcp," in *IEEE INFOCOM*, April 2009, pp. 280–288.

[6] M. Sipos, F. Fitzek, D. Lucani, and M. Pedersen, "Distributed cloud storage using network coding," in *IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2014, pp. 127–132.

[7] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "Xors in the air: Practical wireless network coding," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 3, pp. 497–510, June 2008.

[8] J. Krigslund, J. Hansen, M. Hundebøll, D. Lucani, and F. Fitzek, "CORE: COPE with MORE in Wireless Meshed Networks," in *IEEE Veh. Tech. Conf. (VTC)*, Dresden, Germany, June 2013.

[9] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Internet Requests for Comments, RFC Editor, RFC 6824, January 2013, http://www.rfc-editor.org/rfc/rfc6824.txt.

[10] S. Barré, C. Paasch, and O. Bonaventure, *NETWORKING 2011: 10th International IFIP TC 6 Networking Conference, Valencia, Spain, May 9-13, 2011, Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. MultiPath TCP: From Theory to Practice, pp. 444–457. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20757-0_35

[11] Jana and I. Swett, "Quic: A udp-based secure and reliable transport for http/2," Working Draft, IETF Secretariat, Internet-Draft draft-tsvwg-quic-protocol-00, June 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-tsvwg-quic-protocol-00.txt

[12] Libfuse, https://github.com/libfuse/libfuse, 2017.

[13] R. Stewart and C. Metz, "Sctp: new transport protocol for tcp/ip," *IEEE Internet Computing*, vol. 5, no. 6, pp. 64–69, Nov 2001.

[14] S. Shepler, M. Eisler, and D. Noveck, "Network file system (nfs) version 4 minor version 1 protocol," Internet Requests for Comments, RFC Editor, RFC 5661, January 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5661.txt

[15] R. Tobbicke, "Distributed file systems: focus on andrew file system/distributed file service (afs/dfs)," in *IEEE Symposium on Mass Storage Systems*, 1994, pp. 23–26.

[16] I. Voras and M. Zagar, "Network distributed file system in user space," in *28th Int. Conf. on Info. Tech. Interfaces*, 2006, pp. 669–674.

[17] Y. Hu, C. M. Yu, Y. K. Li, P. P. C. Lee, and J. C. S. Lui, "Ncfs: On the practicality and extensibility of a network-coding-based distributed file system," in *Int. Symp. on Networking Coding*, July 2011, pp. 1–6.

[18] C. W. Sørensen, A. Paramanathan, J. A. Cabrera, M. V. Pedersen, D. E. Lucani, and F. H. P. Fitzek, "Leaner and meaner: Network coding in simd enabled commercial devices," in *2016 IEEE Wireless Communications and Networking Conference*, April 2016, pp. 1–6.

[19] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, Oct 2006.

ISSN (online): 2446-1628
ISBN (online): 978-87-7112-982-3

AALBORG UNIVERSITY PRESS