Application and Implementation of Network Coding for Cooperative Wireless Networks

Pedersen, Morten Videbæk

Publication date:
2012

Document Version
Accepted author manuscript, peer reviewed version

Application and Implementation of Network Coding for Cooperative Wireless Networks
∼ PhD Thesis ∼

Morten Videbæk Pedersen, mvp@es.aau.dk

Application and Implementation of Network Coding for Cooperative Wireless Networks
Ph.D. thesis

ISBN: 978-87-92328-88-5
August 10 2012

Academic advisors:	
Prof. Frank H.P. Fitzek	Aalborg University, Denmark
Prof. Torben Larsen	Aalborg University, Denmark
Internal examiner:	
Assoc. Prof. Jan Østergaard (chairman)	Aalborg University, Denmark
External examiners:	
Prof. Mario Gerla	University of California, Los Angeles
Prof. Michele Zorzi	University of Padova

Abstract

English Abstract

Today the traditional client-server network architecture is the predominant model in our network infrastructure. However, for the increasing amount of "live" services such as TV and radio being digitalized and the growing amount of user generated content, the centralized model can provide a poor utilization of the available network resources. To efficiently support these services we look towards the field of user cooperation. In order to create the incentive for users to join the cooperation we must make the gain larger than the expense. In this PhD I have suggested two central ways of achieving this. 1) I have suggested the use of network coding as a key-enabler for *technology-enabled* cooperation. I refer to technology-enabled cooperation when we are able to provide all participating entities in the network a better performance by enabling user cooperation. In order to achieve this goal I apply network coding, which from a theoretical point of view has the potential to make our networks faster, energy-efficient, robust and more secure. In this PhD I provide an experimental platform for network coding in order to evaluate whether these theoretical merits may be transferred to practice. I provide the initial development of systems and protocols and show that the potential is there. However, I also find that network coding needs to be implemented with care and protocols have to be designed with consideration to make use of this novel technique. 2) The final aspect of this PhD investigates different ways that cooperative models may be implemented to cover a wide range of applications. This addresses the development of user cooperative protocols and how we in Device To Device (D2D) communication may reward users that contribute more to the network than they gain. In this area I suggest the use of social-networks to allow payoff in different domains. In the future this work could be expanded and built into cooperative protocols.

Dansk Resume

I dag er den traditionelle netværksinfrastruktur bygget op omkring en klient-server model. Denne model er dog ikke altid en optimal løsning, for et stigende antal "live" services såsom digital TV og radio. Hertil kommer at brugerne i netværket i lang højere grad end tidligere, også deler data med hinanden. For disse services kan den centraliserede netværksstruktur ofte betyde en dårlig udnyttelse af de til rådighed værende ressourcer. For at effektivisere dette, kan vi finde inspiration indenfor forskningsområdet bruger-samarbejde. Dette kræver dog at vi kan skabe incitament for at den enkelte vil deltage. Dette sker kun hvis vi kan bygge et system hvor fordelene ved samarbejde opvejer ulemperne. I denne PhD har jeg foreslået to centrale måder hvorpå vi kan opnå dette. 1) Jeg har foreslået brugen af netværkskodning, som en nøgleteknologi til at skabe teknologidrevet samarbejde. Jeg referer til teknologidrevet samearbejde når vi er i stand til at forbedre ydelsen for alle brugere der vælger at deltage. For at kunne opnå dette mål udnytter jeg netværkskodning, som i teorien har potentialet til at gøre vores netværk hurtigere, energi effektive, robuste og mere sikre. I denne PhD udvikler jeg en eksperimentel platform hvorfra en evaluering, af disse teoretiske fordele kan undersøges i praksis. Jeg udvikler også første version af systemer og protokoller, og viser potentialet der. Her finder jeg at netværkskodning skal implementeres med omtanke, og at protokollerne skal designes omhyggeligt for at kunne udnytte denne teknik. 2) I den afsluttende del af denne PhD undersøges forskellige måder hvorpå bruger-samarbejde kan implementeres i en lang række applikationer. Dette adressere udviklingen af protokoller til bruger-samarbejde samt hvordan vi i enhed-til-enheds kommunikation kan belønne brugere, som bidrager mere til netværket end de modtagere. I dette område forslår jeg brugen af sociale netværk, til at opbygge en belønningsmodel hvorigennem brugere kan modetage "betaling" for deres deltagelse. I fremtiden kan dette arbejde blive udbygget og inkorporeret i bruger-samarbejdes protokoller.

Preface

This PhD thesis presents a selection of papers which embody the direction and research topics that were investigated throughout my 3 years as a PhD student at the Antennas, Propagation and Radio Networking Group (APNet), Department of Electronic Systems, Aalborg University. This thesis was prepared under the supervision of Professor Frank H.P. Fitzek. This work was financed by the CONE project (Grant No. 09-066549/FTP) granted by the Danish Ministry of Science, Technology and Innovation.

The thesis includes 6 selected publications and complete list of all co-authored publications.

Acknowledgments

First I would like to thank Frank H.P. Fitzek for being the best possible supervisor and great friend. I could not express in words the impact Frank has had on my life since we first met now 6 years ago. Thanks for everything Frank. Also a special thanks goes to Janus Heide with whom I have collaborated closely during both my masters and now also PhD. I also wish to thank Kirsten Nielsen for all her help throughout the project, without her who knows where we would have been. Thanks to Péter Vinglemann for being a great lab buddy during his stay at Aalborg University. Thanks Achuthan Paramanathan, Peyman Pahlevani, Stephan A. Rein and Martin Hundebøll of the mobile devices group for our discussion and collaborations. Thanks to the new generation Jeppe Krigslund and Jeppe Pihl for all their help especially during the last two years. Thanks to Torben Larsen for helping in realizing this project. I would also like to thank Muriel Médard for all her help during this project and for hosting me during my stay with her research group. Thanks to all my colleagues from the APNet section.

Also a big thanks to my friends and family who have supported me throughout all these years. Their encouragements and support have been fantastic and I am truly grateful for it.

The work presented in this thesis is the result of collaborative efforts. So finally I wish to thank everybody how helped me during the process and who made it possible for me to complete this thesis.

Contents

Chapter 1

Introduction

In this chapter we will introduce the main topics and contributions of this PhD thesis and provide the reader with the background and motivation for the work carried out.

In the past few years a remarkable development of commodity mobile communication devices such as smartphones has occurred. This change has not only been on the technological side, where the devices today are as powerful as our desktop computers only a few years ago. But, also in the way that we as consumers have integrated these new devices into our daily lives. Many users already use these devices to store their "digital life" i.e. music, videos, photos, conversations and so forth. But storing content is not the final goal. Users want to share and experience content together. This has already been seen in the remarkable success of on-line services such as Flicker, YouTube, and Facebook. As an example approximated 250 million photos are uploaded to Facebook every day [44]. Although these services provide common storage and distribution functionality, they also introduce an asynchronous content distribution model. Where content is first uploaded and then later at different points in time, downloaded and consumed by other users. This model is a poor fit for building services which provide a "live" experience where users simultaneously enjoy content together with their friends. Reaching this goal, is further complicated by the fact that current transport protocols only provide very limited support for multicast traffic, in which data is delivered to a group of users simultaneously. Also for classical services such as digital TV and radio the traditional client-server model results in a poor utilization of the available network resources. Due to the lack of efficient multicast the same data is copied and transmitted to each user separately. To efficiently support this type of content distribution model we may look towards the field of user cooperation, which breaks with the traditional centralized client-server architecture and allows users to communicate directly.

1.1 User Cooperation in Wireless Networks

In mobile and wireless networks user cooperation refers to the situation where a number of users decide to undertake a certain task as a group rather than as individuals. This fundamental concept has its roots in nature and relies on the belief that through cooperation each individual may achieve its goals spending less resources compared to working alone. In wireless networks these principles are the same, and previous research has demonstrated that the concept has to the potential to improve upon the current systems in a variety of ways [45].

In common for these systems are that they break with the traditional centralized network architecture, as users communicate directly instead of only with a centralized server. A classical example of this is shown in Figure 1.1 where users interested in the same content and within close proximity, establish a secondary communication channel using a short-range communication interface. By doing this the users may improve the performance of the communication by reducing the overall traffic required.

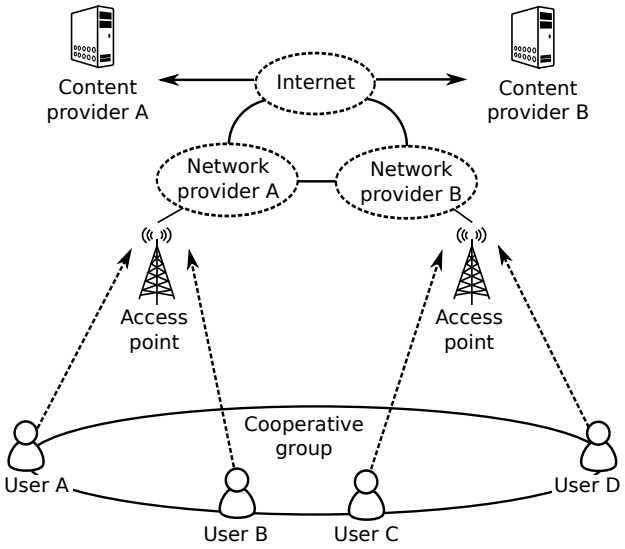


Figure 1.1: Cooperative network architecture, where a number of users form a cooperative cluster in order to efficiently access content via the Internet.

In general the main motivation for introducing user cooperation in a wireless system falls into two categories.

The first category contain systems that *enhance the performance* of the communication system. In this category we find systems that utilize cooperation to improve different performance indicators, typically this could be

energy consumption, throughput or delay. Examples of this are given in [46] where user cooperation combined with Multiple Description Coding (MDC) is used to lower the resources needed to deliver a live video to a group of users. In [47] the authors improve the throughput of mobile web browsing by accumulating the cellular capacity of the individual users into one big virtual data-pipe.

The second category contain systems that *enhance the functionality* of the communication system. In this category systems utilize user cooperation to provide functionality to the participating users not otherwise available. This could be digital resources such as photos, audio and video. But also physical resources such as cameras, sensors, etc.

Although research has shown many benefits of user cooperative techniques in both theory and implementation, only few examples of user cooperation in communication networks exists in actual products and systems. Some of the most successful are the now quite popular wireless hot-spot solutions where an user can turn his or her smart-phone into a mobile access-point and thereby share the cellular connectivity with other users in close proximity [48]. Although this serves as a nice example of user cooperation the potential is still underutilized. We believe that this in part can be explained by the missing solutions to some of the following challenges.

Traditionally user cooperation has been systems oriented, in this case the benefit of the individual devices or users are less important than the system as a whole. For this type of cooperation the success criteria is that the overall system gains by the user cooperation. Although this might work well in fully dedicated networks such as sensor- or mesh-networks where the system controls and "owns" all the participating devices. It does not fit well into the mobile end-user networks. In these networks devices are owned by selfish individuals, who are unlikely to sacrifice resources to improve performance for a complete stranger. Therefore we have to either build systems where all users will benefit from cooperation or find alternative ways so that users who sacrifices resources in one domain may receive compensation or rewards in another domain.

In the protocol domain a different challenge when building systems relying on user cooperation is that the *complexity* of the systems grow significantly as the number of cooperating users increase [49, 50]. This if further complicated by the dynamic and unstable nature of mobile networks which mean that maintaining a consistent protocol state and determining which users should cooperate becomes increasingly difficult. As a simple example consider the cooperative data distribution network shown in Figure 1.2. In order to minimize the traffic required from the server, the access point will only transmit data until the users combined have the full information. However

due to transmission errors it is likely that all nodes after some time only have partially filled buffers. As the users receive data via the cellular link they utilize their short-range network interfaces to exchange their missing packets.

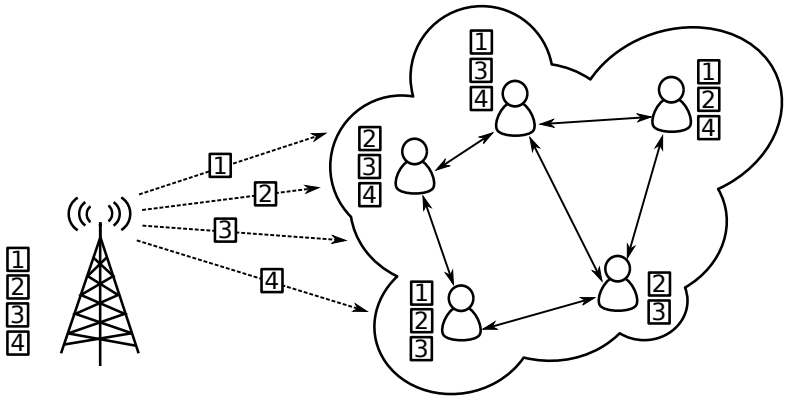


Figure 1.2: The scheduling problem with user cooperation where each receiver holds a different set of packets (denoted by numbers). No single packet can benefit all receivers and the repair phase therefore becomes suboptimal.

As seen in Figure 1.2 not one single packet is useful for all receivers and the task of understanding which users require which packets now becomes a significant challenge.

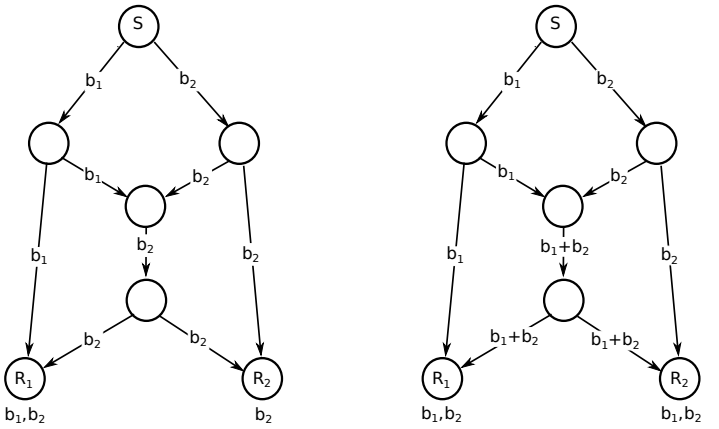
As illustrated here deploying user cooperation in current and future networks will require a number of new solutions to address the existing challenges. However, in 2000 it seemed as if a piece of the puzzle was found as Ahlswede et al. introduced the theory of network coding [51]. Investigating the cross-over between network coding and user cooperation was the starting point for this PhD work.

1.2 Network Coding a Key Enabler for User Cooperation

Network coding has the potential to improve the data distribution among the users participating in the cooperative network. In the following we will introduce network coding and underline its benefits towards user cooperation.

Network coding breaks with the traditional paradigm in packet switched networks often referred to as *store-and-forward*. In this type of network nodes on the intermediate path of a packet flow simply receives and forwards the incoming packets without performing any kind of processing of the packets. In network coding packet flows are no longer considered immutable entities

and intermediate nodes in the network may choose to *recode* packets before forwarding them. Due to this unique feature networks utilizing network coding are often referred to as *compute-and-forward*. A classical example of how network coding changes the way data is processed in the network is illustrated by the famous Butterfly example shown in Figure 1.3.



(a) The *store-and-forward* approach from traditional routing.

(b) The *compute-and-forward* approach from network coding.

Figure 1.3: The Butterfly network in which each link can carry one packet per unit time. (a) Shows the traditional solution where nodes simply forwards the incoming packets. (b) Shows the network coding approach where the bottleneck node codes the two packets b_1 and b_2 into one coded packet $b_1 + b_2$ in this case + represents the addition in a Finite Field.

On the left-hand side (Figure 1.3a) we see how using traditional routing we may deliver on average 1.5 packets per time unit at the two receivers. Whereas on the right-hand side (Figure 1.3b) we see that the bottleneck node is "allowed" to recode the two packets b_1 and b_2. Consequently the two receivers are able to successfully decode both packets. The Butterfly elegantly illustrates the key operation of network coding. Since the introduction of network coding significant efforts have been invested trying to understand the implications of this new technique and how this seemingly simple idea could be transferred to communication networks in practice. In the context of user cooperation one of the most significant contributions to this were introduced in [52] where the authors showed that for a multicast transmission, randomly creating linear combinations of the incoming data packets over a sufficient large finite field were enough to ensure a close to optimal performance. This

approach was named Random Linear Network Coding (RLNC). Figure 1.4 depicts the basic operations of RLNC. To lower the computational complexity large files are typically split into several equally sized chunks, called *generations*, each generation then consists of g packets.

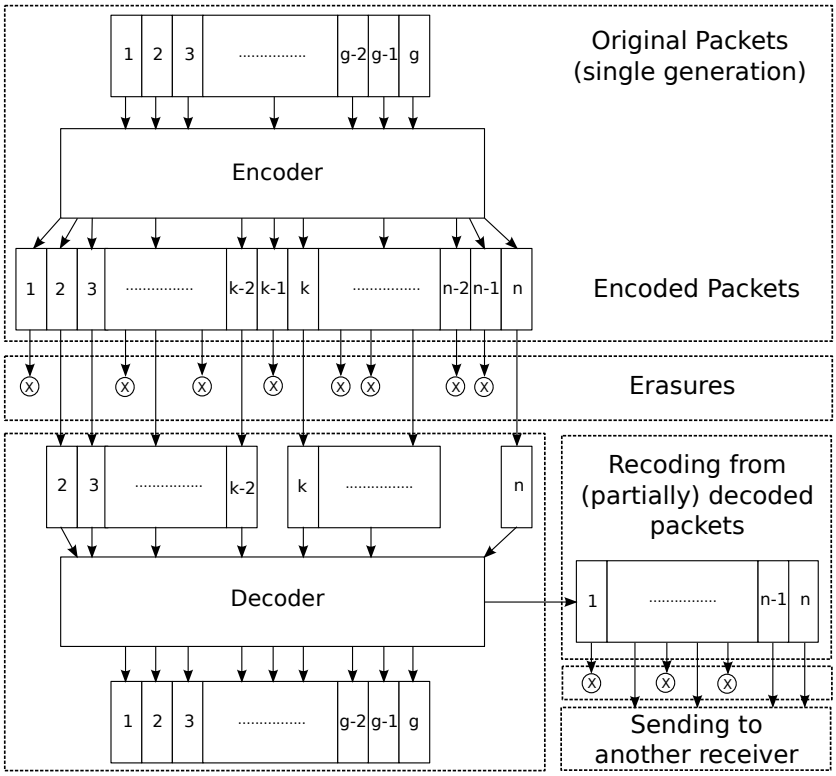


Figure 1.4: RLNC system overview showing the encoding, decoding and recoding operations.

The encoder (shown in the top of Figure 1.4) generates and transmits random linear combinations from the data packets of the current generation. The linear combinations are created over the chosen finite field. With RLNC the coding coefficients are chosen randomly which means that any number of encoded packets can be generated from any given generation. The middle layer represents the wireless channel, where packets maybe lost depending on the channel conditions. At the receivers packets are passed to the decoder (the bottom component of Figure 1.4), which will be able to reconstruct the original data packets after receiving g linear independent combinations. Finally a receiver may choose to generate and send new encoded packets based on the currently partially decoded packets. This operation is known as recoding and is the unique feature of network coding.

This approach looks promising for user cooperation since it relies on a very decentralized approach with very limited need for coordination between cooperating users. Due to these proprieties RLNC was by many seen as a very interesting tool for many different types of wireless networks, where the unreliable nature of wireless in many cases favors decentralized algorithms with minimum coordination requirements [53, 54]. However, in spite of the nice theoretical properties a concern was whether the network coding algorithms were too complex for even modern day desktop computers and mobile devices [55, 1]. This raised the question whether the use of network coding would add too much computational complexity and that the user cooperation would lose its benefit over traditional client-server networking.

1.3 Thesis Outline

In the previous sections we have introduced the purpose of utilizing user cooperation and network coding and outlined some of the challenges faced by the research community in trying to makes these techniques applicable and useful in actual wireless networks. In the following we will introduce the two core aspects presented in this PhD thesis.

1. Implementation of network coding algorithms.
2. Integration of user cooperation and network coding.

1.3.1 Implementation of Network Coding Algorithms

One of the main challenges in the initial phase of the PhD work was the lack of a suitable experimental platform for network coding algorithms. At the time network coding remained a largely theoretical field and several researchers voiced their concerns over the complexity of network coding being too high to be practical in real networks [55]. The need to begin development of a suitable platform for experimentation with network coding was therefore outspoken. The goal was that this development effort should result in a better understanding of how the reported high complexity would translate into actual performance on state-of-the-art consumer available hardware. Modern hardware has become extremely complex and in order to tune an implementation to get the best possible performance it requires an understanding of many optimization aspects such as vectorization using Single Instruction Multiple Data (SIMD), memory access patterns and caches, assembly instruction latencies, etc [56, 57, 58]. The possibility to apply these optimizations in many case depend on the algorithms design and structure, therefore having

an available implementation could be used to provide valuable feedback to researchers working on techniques for lowering the complexity of the suggested algorithms. In effect providing the missing link in the optimization cycle shown in Figure 1.5.

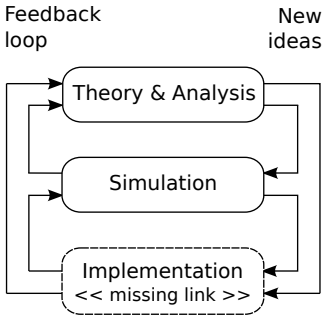


Figure 1.5: Optimization cycle of creating practical network coding algorithms.

To achieve this a number of goals were specified for the development of the library, the initial goals are presented in [7]. These have since been updated, however the essence remains the same namely to create a design which allowed easy modification and experimentation while at the same time yielding highly optimized implementations.

The development of a suitable software platform for experimentation has been an ongoing activity throughout the entire PhD period. The main result of this effort were two libraries written in C++ to allow both efficient and wide platform support. The first library called *Fifi* implements the mathematical operations needed by the network coding algorithms. The arithmetic operations used are defined within a branch of mathematics known as Finite Fields or Galois Fields. In network coding finite field arithmetics are used when performing the three core operations namely: encoding, recoding and decoding. An efficient implementation of finite field arithmetics is therefore an important prerequisite for any network coding implementation. At the time of writing we believe that Fifi provides one of the most comprehensive implementations of finite field arithmetics currently supporting the following fields and algorithms.

- *SimpleOnline*$\{8, 16\}$: This algorithm computes the result on-the-fly in $\mathbb{F}_{2^8}$ using an iterative algorithm, without any precomputed lookup table. The *SimpleOnline* algorithm supports the $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$ field.
- *OptimalPrime2325*: This algorithm presents an alternative to the traditional binary extension fields. Using the prime field $\mathbb{F}_{4294967291}$, where $p = 2^{32} - 5 = 4294967291$

- *FullTable8*: This algorithm utilizes a fully precomputed lookup table stored in memory to calculate the results in $\mathbb{F}_{2^8}$.

- *LogTable*$\{8, 16\}$: This algorithm uses a reduced lookup table to calculate the results in $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$. The log table minimize memory consumption at the cost of additional operations for every calculation.

- *ExtendedLogTable*$\{8, 16\}$: This algorithm extends the lookup table used by the *LogTable* to calculate the results in $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$. The extended lookup table removes a number of checks necessary in the *LogTable* algorithm when moving from exponential to polynomial representation.

In [31] we present our work on implementing the Finite Field arithmetics used by network coding algorithms. We provide implementers with guidelines for choosing between the different finite field algorithms.

In [2] we introduce the use of an Optimal Prime Field for finite field arithmetics. Optimal Prime Fields utilizes higher order prime fields instead of the binary or binary extension field typically used. The main advantage of the Optimal Prime Field is the larger field size and efficient implementation. However, also several drawbacks exist which are discussed in the paper.

Built on-top of Fifi is the *Kodo* library. Kodo implements a selected set of the network coding algorithms proposed in literature. Kodo was designed utilizing a specific C++ design technique proposed by [59, 60] called Mixin-Layers. Mixin-Layers offer a very generic approach to structuring a software library. The end result is a large set of building blocks rather than a fixed set functionality, these building blocks can then be assembled at compile time to yield desired functionality, at the same time as giving the compiler the full static information about the composed blocks. This allows the compiler to emit machine code equivalent to a hand-written special purpose function. While also giving the developer, in this case the researcher experimenting with different algorithms, the freedom to easily compose and customize the available functionality. The use of Mixin-Layers has previously been successfully applied to high-performance memory allocators [61]. It is to the best of our know the first time this technique is applied for error-correcting codes.

In [7] we present the first open source version of the library. The paper presents the initial design and gives examples on how the library can be used to implement network coding algorithms.

1.3.2 Integration of Network Coding and User Cooperation

In this part of the PhD work we begin investigating the cross-over between user cooperation and network coding.

As we have previously mentioned the integration of user cooperation in mobile end-user networks are particular challenging due to the fact that most users will not act purely altruistic. Although the system i.e. the network operator and the cooperating users as a whole might gain from the cooperation. The selfish user will not participate if there is not an individual gain for him or her. Here we do not consider *forced cooperation* although this might be a plausible scenario if the user cooperation is built into the network technology or if the network operator has access to control the users devices. This means that the incentive to join the cooperation is based purely on egoistic behavior and will only take place if the user sees a gain. If the user sees an advantage to cooperate due to e.g. better date rate or lower energy consumption we call this *technology enabled cooperation*. The integration of network coding with user cooperation has the purpose of making the user cooperation more effective and thereby increase the willingness of users to join the technology enable cooperation. Whether this will be the case in practical systems relies on several factors. 1) How does the added complexity of the network coding algorithms affect the performance of the wireless protocol. 2) Can we build protocols which utilize the special "recoding" properties of network coding to increase performance.

In [3] we investigate the effect of network coding on standard mobile devices. We show that utilizing network coding has a measurable impact on the throughput of the device. Continuing the effort to keep improving the network coding algorithms is therefore important. We provide an initial investigation of a network coding based protocol. These investigations shows that for the given topology the protocols should be able to tune their activities depending on the Packet Erasure Probability (PEP) on the network. Also in the specific setup the use of small generation sizes showed a higher energy-per-bit usage due to the wasted linear dependent transmissions, whereas large generations sizes had an increased energy-per-bit due to the increased computational load.

In [8] we introduce a practical protocol to facilitate the dissemination of multimedia towards a cooperative cluster of smartphones. The protocol uses RLNC to increase the efficiency of the communication within the cooperative cluster. In order to evaluate the protocol a test-bed was created and the several measurements were carried out. The evaluation showed that ENOC Cooperation Protocol (ECP) was capable of recovering close to the maximum

amount of packet errors. Furthermore, the evaluation confirmed that in a practical setting the field size should be carefully chosen to avoid overloading the resource constrained devices, and thereby increasing the possibility of buffer overflows etc.

The development of systems based on technology enabled cooperation has a wide potential. However, there exists systems where the technology enabled cooperation model typically does not apply and a different cooperation model is therefore needed. In this model the service typically consist of a service provider i.e. one who "contributes" a resource to the system. This could be the user who grants neighboring devices access to his or her Internet connectivity. In current systems this model has been based on altruistic behavior, i.e. typically the "contributor" has some sort of relationship with the receivers, which outweighs the fact that he or she will spend resources without payback. In some cases the contributor might gain in terms of strengthening the personal relationship with the receivers or rise in reputation or esteem. To enable this type of cooperative model we may capitalize on the widespread popularity of online social networks.

In [4] we introduce the concept of the "mobile cloud" as an alternative to the existing client-server architecture in most content distribution networks. The mobile cloud enhances the existing communication architecture utilizing user cooperation, Device To Device (D2D) communication and network coding. This results in a more content centric distribution model where the users both host and participate in the content distribution. In order to motivate and increase the users willingness to participate in the cooperative distribution socially enabled cooperation is introduced. In this model users participating in the cooperation and sacrificing resources may be rewarded though their social network.

Chapter 2

Contributions In This Thesis

In this chapter we present the publications included in this thesis. Where relevant we also point the reader to "own related publications". These publications were also part of the PhD work but have not been included in the thesis.

2.1 Paper 1

Mobile Clouds: The New Content Distribution Platform
Morten V. Pedersen, and Frank H.P. Fitzek
Institute of Electrical and Electronics Engineers. Proceedings, Vol. 100, 13.05.2012.
Pages 4.

Motivation

In recent years the success of online content sharing services and social networks have changed the way that users interact and share content over the Internet. This means moving away from a model where content is mainly produced by "classical" publishers such as news and television networks, to a model where users to a large extend contribute and share the content over the Internet. In this new model the question is whether the classical client-server network architecture still offers the best solution. Or whether it should be changed to more efficiently support this new more decentralized content distribution pattern.

Paper Content

The paper introduces the current problems and motivation for enhancing the traditional client-server network architecture. This is achieved by utilizing ideas from the field of user cooperation [62]. Based on user cooperation a more decentralized content distribution model utilizing D2D communication is suggested. The success of such a model will depend largely on the willingness of the users to participate in the cooperation. Different cooperation incentives are therefore introduced and discussed.

Main Results

The paper introduces the concept of the "mobile cloud" as an alternative to the traditional client-server network architecture. The "mobile cloud" is envisioned to enhance the existing networks by combining techniques from user-cooperation, D2D communication and network coding. Dealing with the users willingness to cooperate is a critical aspect in the foundation for any cooperation based system. We therefore suggest the concept of "socially enabled" cooperation. Socially enabled cooperation proliferates on the success of online social networks to create an different way of rewarding users willing to cooperate.

Own Related Publications

In [5] we extend the work presented here on the integration of social and cooperative networks. Furthermore we elaborate on the importance of utilizing network coding in the "mobile cloud" when building user cooperative protocols.

2.2 Paper 2

On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices
Péter Vingelmann, Morten Videbæk Pedersen, Frank H. P. Fitzek and Janus Heide
Paper presented at Globecom 2011, Houston
Pages 6.

Motivation

Today most mobile devices have several available communication interfaces i.e. most smartphones contain both a cellular and often several short-range interfaces. This makes it possible to maintain multiple parallel communication sessions, where one or more network interfaces are used to enhance the performance of the ongoing communication. An example of this could be several mobile users watching mobile TV using their cellular network interface, while at the same time being connected to the same local network using a short-range network interface. As the users typically experience different packet losses, the local short-range network could be utilized as a secondary repair channel. In this case users would cooperate to reduce the amount of redundancy needed on the cellular network. Moving traffic load from the cellular to the short-range network is often advantageous since the short-range network technologies typically provide higher throughput, lower delay, lower energy-per-bit when compared to the cellular technologies.

Paper Content

This paper investigate the possibility of packet error recovery in a cooperative cluster of mobile devices. We assume that these devices receive data from a broadcast transmission on their primary network. Using a secondary short-range network they form a cooperation cluster in order to exchange missing data packets. To achieve this goal ECP is described and implemented. ECP describes a number of mechanisms which are used to make the cooperative exchange as efficient as possible. Following this ECP is implemented in a test-bed using smartphones and Wireless Local Area Network (WLAN) as short-range technology. Using the test-bed a cellular broadcast transmission is emulated and the performance of ECP is measured on the short-range network.

Main Results

In this paper we have introduced a practical protocol to facilitate the dissemination of multimedia towards a cooperative cluster of smartphones. The protocol uses RLNC to increase the efficiency of the communication within the cooperative cluster. In order to evaluate the protocol a test-bed was created and the several measurements were carried out. The evaluation showed that ECP was capable of recovering close to the maximum amount of packet errors. Furthermore, the evaluation confirmed that in a practical setting the field size should be carefully chosen to avoid overloading the resource constrained devices, and thereby increasing the possibility of buffer overflows etc.

Own Related Publications

In [9] we investigate the potential gain from using user cooperation and network coding in existing Long Term Evolution (LTE) networks. LTE networks are supposed to use Forward Error Correction (FEC) codes for the content distribution such as download and streaming services over the air towards the mobile device. In order to minimize the required redundancy by the FEC code it is proposed that local retransmissions using network coding can be used. The proposed approach shows that local retransmissions can save up to 80% of the redundant information on the cellular link as long as there are at least two cooperative users. The result also show that by using network coding the traffic on the short-range network can be reduced by 50% as long as there are four devices in the cooperative cluster.
In [10] we extend previous work to include a high degree of mobility. In this setting the cooperative cluster is subject to sporadic disconnections and only partial connectivity as devices move in and out of range. The paper compares the use of a network coding based User Datagram Protocol (UDP) protocol and a reference scheme based on Transmission Control Protocol (TCP). Results show that the proposed strategy is able to outperform the reference strategy both in terms of good-put and energy consumption.

2.3 Paper 3

Kodo: An Open and Research Oriented Network Coding Library
Morten V. Pedersen, Janus Heide, and Frank H.P. Fitzek
NETWORKING 2011 Workshops: International IFIP TC 6 Workshops, PE-CRN, NC-Pro, WCNS, and SUNSET 2011, Held at NETWORKING 2011, Valencia, Spain, May 13, 2011, Revised Selected Papers. Vol. 6827 Springer, 2011. p. 145-153 (Lecture Notes in Computer Science).
Pages 8.

Motivation

Since the introduction of network coding in 2000 by Ahlswede [51] much work has been carried out showing the theoretical benefits of this novel new approach to data distribution. However, since then only a limited amount of work has been carried out investigating the feasibility of these algorithms in practice. In this paper we introduce a high performance research oriented C++ library targeting researchers and practitioners wishing to work on practical network coding. It is the hope that this library may serve as a starting point for researchers wishing to investigates the practical aspects of network coding.

Paper Content

The paper introduces design goals and motivation behind the Kodo library. Following this the paper provides an overview of different network coding approaches and provides a description of the network coding algorithms supported by Kodo. The initial functionality covers most of required algorithms for implementing a RLNC based system. The paper also introduces how Kodo handles packetization through the use of partitioning schemes. Finally an example shows how to use the basic RLNC classes to perform encoding and decoding of a data block. The example also shows how to use different density generators for the encoding vectors and allows changing the used finite field.

Main Results

The goal of this paper is to provide other researchers an way to experiment with practical network coding algorithms. This goal has been so far been successfully achieved and the library has been reported used at several research institutions worldwide. The library supports implementing an operational

RLNC application in only 60 lines of C++ code. Furthermore the code has been successfully used on the following platforms Windows, Linux, Mac OS and Android.

Own Related Publications

In [11] we consider different approaches to reduce the number of operations needed by the decoding algorithms in RLNC. In particular we are interested in the case where the coding vectors are sparse. We use an on-the-fly version of the Gauss-Jordan algorithm as the baseline, and provide several simple improvements to reduce the number of operations needed to perform decoding. Our tests show that the improvements can reduced the number of operations needed with 10-20% on average depending on the encoding parameters.

2.4 Paper 4

Finite Field Arithmetics for Network Coding
Morten Videbæk Pedersen, Janus Heide and Frank H. P. Fitzek
Book chapter in "Network Coding: A Hands-on Approach" tentative title, to be published Wiley.
Pages 39.

Motivation

Finite Fields or Galois Fields are the underlying mathematical foundation of network coding algorithms. In practice we may choose between a wide range of different field implementations and realizations. Choosing a specific implementation can depend on several factors. Some choices are dictated by topology i.e. in order to achieve the multicast capacity of certain networks theory tells us what is the required field size [63]. On the other hand practical concerns also limits our freedom of choice. Typically choosing a large field will increase the complexity of the mathematical algorithms. In this book chapter we investigate the implementation of different Finite Fields and their impact on network coding algorithms.

Paper Content

The chapter starts with a brief introduction to the theory of finite fields. Following this the chapter introduces how the theory can be transformed into software algorithms. The chapter starts at the binary field and shows how the field size may be increased through the use of binary extension fields. For the binary extension fields a number of different implementation techniques are presented and discussed. The algorithms required for the implementations are also presented and discussed. Furthermore the trade-off between the different algorithms in terms of complexity and memory consumption is shown.

Main Results

The results presented in this chapter makes it possible for network coding researchers to include practical concerns when choosing the type of finite field to used. It also makes available to practitioners the algorithms and descriptions needed to implement the finite fields in network coding systems. The paper demonstrates how to transform the mathematical constructs into

runnable algorithms. The paper introduces 5 different algorithms for implementing finite field arithmetic. For all algorithms memory consumption and example implementations are shown and described.

Own Related Publications

In [12] we investigate the trade-off between key parameters in a network coding system namely field size, generation size, coding vector density and coding vector representation. We show that for a simple topology a low field size offers the best trade-off.

2.5 Paper 5

Network Coding Over The $2^{32} - 5$ Prime Field
Morten Videbæk Pedersen, Janus Heide, Péter Vingelmann and Frank H. P. Fitzek
IEEE Transactions on Mobile Computing (*in preparation for submission*)
Pages 9.

Motivation

From theory it can be shown that increasing the field size used in a network coding system increases the efficiency of the code as it reduces the probability of transmitting linear dependent packets in the network. In certain cases a high field size may even be required in order to realize the communication's maximum theoretical data rate. However, in practice implementations typically search to use the smallest possible field size as this in most cases is easier to implement efficiently and therefore yields a higher performance. There is a continued need to find better and more efficient ways of implementing finite field algorithms.

Paper Content

The paper proposes the use of a finite field called optimal prime fields for network coding systems. In order to provide a practical solution the paper first introduces solutions to two obstacles. Namely efficient implementation of the modulo operation and mapping arbitrary binary data to the selected field. Following this several approaches to the binary mapping are investigated and their practical performance is measured. Following this the performance of the proposed field is evaluated and compared to different field implementations. Finally the paper provides a discussion and conclusion of the proposed solution.

Main Results

The main result found shows that the optimal prime field is a promising candidate to implement higher order fields in network coding systems. The performance is between 18% and 20% faster than the currently fastest F_{2^8} fields implementation tested, while at the same time providing a significant higher order field, namely with $2^{32} - 5$ field elements. Another advantage of the proposed solution is that the computations does not rely on any precom-

puted look-up tables or similar. Which makes it a suitable candidate for low memory devices such as sensors etc.

2.6 Paper 6

A mobile application prototype using network coding
Morten Videbæk Pedersen, Janus Heide, Frank H. P. Fitzek and Torben Larsen
European Transactions on Telecommunications, Vol. 21, No. 8, 12.2010, p. 738-749.
Pages 12.

Motivation

Several open questions exists when considering the use of network coding combined with user cooperation. One is whether the computational complexity added by the additional network coding operations surpasses the gains obtained. Furthermore following the implementation of network coding algorithms is the implementation of network coding protocols. Protocol designers should begin to consider how this new technique can be applied to communication protocols in an efficient and meaningful way.

Paper Content

The paper introduces a simple single-hop cooperative network and provide an overview and comparison of the different data distribution techniques. Following this the network coding algorithms used are introduced and implemented on a smartphone based test-bed. Using the test-bed a number of measurements are conducted in order to measure the impact of the coding operations on the network performance. The measurements are based on a simple setup where a source transmits a chunk of data to all receivers. To achieve this in the most efficient way the use of network coding is evaluated in two steps. First network coding is used as a traditional FEC code with no recoding and communication between the receivers. In the second step the receiving devices participate by transmitting recoded packets. Based on these two schemes several measurements are recorded and the performance of the two schemes are evaluated. Based on the observed results a number of recommendations and considerations for future network coding based protocols are presented.

Main Results

Utilizing network coding has a measurable impact on the throughput of the device, continuing the effort to keep improving the network coding algorithms

are therefore important. In the specific setup recoding should be used carefully, taking into account the packet error probability of the receivers is a first guideline for determining when a receiver becomes a useful relay. The additional linear dependency introduced by the binary field creates a trade-off between generation size and energy consumption. For small generation sizes a higher field size could yield better performance, this is however subject to further investigations.

Own Related Publications

In [13] we present an analysis of binary field algorithms used in this paper. We propose the use of the binary finite field to increase performance and present an evaluation of the proposed solution. We also quantify the speed-up from using a systematic phase at the beginning of the FEC block transmission.
In [14] the initial prototype application is developed and the first implementation results are presented. The prototype is able to demonstrate the use of network coding on resource constrained mobile devices. A simple protocol based on a no feedback packet overshoot scheme is presented and used to control the data transmission.
In [15] a number of additional platforms are included. This is done to quantify the performance of an extended set of hardware architectures. The platforms include the popular iOS based devices.

Chapter 3

Contributions of the PhD Work

This chapter summarizes the main contributions of the PhD work. In the following we will describe the main outlets of the knowledge generated throughout the project and describe some of the initiatives that were started throughout of the PhD.

The main instruments for dissemination in this project has been through a) *publications*, b) *open source software*, c) *demonstrators*, d) *teaching*, e) *organization of workshops*, f) *research projects*, g) *start-up company*.

a) As a part of the PhD project one of the main dissemination channels have been through publications at conferences and workshops. At the time of writing this has resulted in 15 co-authored papers (2 as first author). In addition to this a number of journal and book chapters has been co-authored which currently counts 5 journal papers (3 as first author) and 4 book chapters (2 as first author). For the full list of publications see the "Complete List of Publications" included in Chapter 5.

b) During the project period a large amount of research based software was developed. This software has been made publicly available on the Internet to other researchers and students working on related subjects. The source code for the two main projects *Fifi* and *Kodo* can be found here:

   - https://github.com/steinwurf/fifi
   - https://github.com/steinwurf/kodo

c) Throughout the PhD project a number of demonstrators has been developed. These demonstrators have been instrumental in communicating the concepts of user cooperation and network coding to wider audience. They also serve as clear way to demonstrate the potential of user cooperation and network coding. One example of this was a demonstrator

developed in collaboration with Prof. Muriel Medard's research group at Massachusetts Institute of Technology (MIT) showing network coding used to support the distribution of live video streams. This demonstrator was shown at National Broadcasting Company (NBC), to demonstrate how user cooperation and network coding could be implemented into future video distribution networks.

d) During the project period a large amount of teaching activities have been carried out. During the PhD project we have applied much of our research to mobile phones and we believe the mobile phone serves as an excellent platform for research, experimentation and demonstration. In order to help others get started with these platforms we have organized summer schools in mobile phone programming in 2010, 2011, and 2012. Each year with an average of 20 to 30 participants from all over Europe. In addition to this we have given a number of smaller lectures on mobile phone development for a variety of different people from 9th grade school kids to company employees.

e) During the PhD it has been a pleasure to server in the Technical Program Committee (TPC) and support in organization of the ICC CoCoNet 4 and ICC CoCoNet 5 workshops on cognitive and cooperative wireless networks.

f) The Evolved Network COding (ENOC) project was started in cooperation with and funded by Nokia. The main scope was to investigate cooperation and network coding in cellular networks, and to produce publications and patents on the topic. Later this project was continued as Network COding Evolved (NOCE) in cooperation with Renesas Mobile. Finally I have received the funding to continue the work started during this PhD by the Danish Ministry of Science, Technology and Innovation as a three year individual Post Doc.

g) In 2011 Steinwurf ApS was founded by Janus Heide, Frank H.P. Fitzek, Muriel Médard and Morten V. Pedersen. The company will deliver software applications and protocols to customers who wish to incorporate user cooperation and network coding as part of their network infrastructure. The company is currently in the start-up phase and has received the initial seed funding in 2011.

Chapter 4

Conclusion

Throughout this PhD we have been investigating the cross-over between user cooperation and network coding, as a way to enhance the data dissemination in mobile networks. Our initial starting point was the development of an experimental platform for network coding based protocols. These efforts have been open-sourced and are today freely available to researchers working on practical network coding. The developed platform has since been used in a series of publications to investigate to what degree the reported high complexity of network coding would impact practical systems and how it potentially could be reduced. Two areas significantly contribute to the computational complexity of the algorithms, namely the finite field arithmetics and the choice of parameters for the encoding and decoding algorithms. In this thesis we present our work on lowering the complexity of the finite field arithmetics. One promising technique is the use of the Optimal Prime Field which efficiently utilizes normal integer arithmetics of the Central Processing Unit (CPU) and thereby offers fast calculations. In the related work, we reference the work we have done on reducing complexity of the decoding algorithms [11]. Based on the developed platform we investigate the impact and use of network coding in user cooperative protocols. We show that user cooperation can benefit from the use of network coding and that although the complexity of network coding does impact the performance it still surpasses the performance of state-of-the-art reference schemes. However, we also see that even for simple single-hop cooperative clusters the protocols have to be carefully designed to efficiently take advantage of network coding. Finally we present our work on creating cooperation incentives in networks where the users are not contributing equally to the network. To make cooperation attractive in such networks we propose the use of social networks to create an alternative payoff model.

Within all of these areas we are by no means at the end of the road

but we believe that the contributions from this project will serve as a useful foundation and input for researchers also working in the field.

Chapter 5

Complete List of Publications

Journal Papers

[1] Janus Heide Møller, Morten Videbæk Pedersen, Frank H.P. Fitzek, and Torben Larsen. "Cautious View on Network Coding - From Theory to Practice". In: *Journal of Communications and Networks* 10.4 (2008), pp. 403–411. ISSN: 1229-2370.

[2] Morten Videbæk Pedersen, Janus Heide, Peter Vingelmann, and Frank H.P. Fitzek. "Network Coding Over The 232-5 Prime Field". In: *IEEE Transaction on Mobile Communication* (2012). In preparation for submission.

[3] Morten Videbæk Pedersen, Janus Heide, Frank H.P. Fitzek, and Torben Larsen. "A Mobile Application Prototype using Network Coding". In: *European Transactions on Telecommunications* 21.8 (2010), pp. 738–749. ISSN: 1124-318X.

[4] Morten Videbæk Pedersen and Frank H.P. Fitzek. "Mobile Clouds: The New Content Distribution Platform". In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1400 –1403. ISSN: 0018-9219.

[5] Frank H.P. Fitzek, Janus Heide, Morten Videbæk Pedersen, and Marcos Katz. "Implementation of Network Coding for Social Mobile Clouds". In: *IEEE Signal Processing Magazine* (2012). Accepted. ISSN: 1053-5888.

[6] Peter Vingelmann, Frank H.P. Fitzek, Morten Videbæk Pedersen, Janus Heide, and Hassan Charaf. "Synchronized Multimedia Streaming on the iPhone Platform with Network Coding". In: *IEEE Communications Magazine* 49.6 (2011), pp. 126–132. ISSN: 0163-6804.

Conference Proceedings

[7] Morten Videbæk Pedersen, Janus Heide, and Frank Fitzek. "Kodo: An Open and Research Oriented Network Coding Library". In: *NETWORKING 2011 Workshops*. Vol. 6827. Lecture Notes in Computer Science. Springer, 2011, pp. 145–153. ISBN: 978-3-642-23040-0.

[8] Peter Vingelmann, Janus Heide, Morten Videbæk Pedersen, and Frank Fitzek. *On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices*. ConferencePaper. Paper presented at Globecom 2011, Houston, Texas, USA. 2011.

[9] Qi Zhang, Janus Heide, Morten Videbæk Pedersen, and Frank Fitzek. *User Cooperation with Network Coding for MBMS*. ConferencePaper. Paper presented at Globecom 2011, Houston, Texas, USA. 2011.

[10] Peter Vingelmann, Morten Videbæk Pedersen, Janus Heide, Qi Zhang, and Frank Fitzek. *Data Dissemination in the Wild: A Testbed for High-Mobility MANETs*. IEEE ICC 2012 - Ad-hoc and Sensor Networking Symposium. 2012.

[11] Janus Heide, Morten Videbæk Pedersen, and Frank Fitzek. "Decoding Algorithms for Random Linear Network Codes". In: *NETWORKING 2011 Workshops*. Vol. 6827. Lecture Notes in Computer Science. Springer, 2011, pp. 129–137. ISBN: 978-3-642-23040-0.

[12] Janus Heide, Morten Videbæk Pedersen, Frank Fitzek, and Muriel Medard. "On Code Parameters and Coding Vector Representation for Practical RLNC". In: *IEEE International Conference on Communications* (2011), pp. 1 –5. ISSN: 1550-3607.

[13] Janus Heide, Morten Videbæk Pedersen, Frank H.P. Fitzek, and Torben Larsen. "Network Coding for Mobile Devices - Systematic Binary Random Rateless Codes". In: *The IEEE International Conference on Communications (ICC)*. Dresden, Germany, 2009.

[14] Morten Videbæk Pedersen, Janus Heide, Frank H.P. Fitzek, and Torben Larsen. "PictureViewer - A Mobile Application using Network Coding". In: *The 15th European Wireless Conference (EW)*. Aalborg, Denmark, 2009.

[15] Morten Videbæk Pedersen, Janus Heide, Peter Vingelmann, Laszlo Blazovics, and Frank Fitzek. "Multimedia Cross–Platform Content Distribution for Mobile Peer–to–Peer Networks using Network Coding". In: *Proceedings of the international conference on Multimedia, MM'10*.

2010; 1094. Association for Computing Machinery, 2010, p. 1091. ISBN: 978-1-60558-933-6.

[16] Yao Li, Peter Vingelmann, Morten Videbæk Pedersen, and Emina Soljanin. "Round-Robin Streaming with Generations". In: *International Symposium on Network Coding*. Cambridge, USA, 2012.

[17] Janus Heide, Peter Vingelmann, Morten Videbæk Pedersen, Qi Zhang, and Frank Fitzek. *The Impact of Packet Loss Behavior in 802.11 b/g on the Cooperation Gain in Reliable Multicast*. Paper presented at 76th IEEE Vehicular Technology Conference, Quebec, Canada. 2012.

[18] Martin Hundebøll, Jeppe Leddet-Pedersen, Janus Heide, Morten Videbæk Pedersen, Stephan Alexander Rein, and Frank Fitzek. *CATWOMAN: Implementation and Performance Evaluation of IEEE 802.11 based Multi-Hop Networks using Network Coding*. Paper presented at 76th IEEE Vehicular Technology Conference, Quebec, Canada. 2012.

[19] Janus Heide, Qi Zhang, Morten Videbæk Pedersen, and Frank Fitzek. *Reducing Computational Overhead of Network Coding with Intrinsic Information Conveying*. VTC Fall 2011, San Fransico, USA.

[20] Frank Fitzek, Janus Heide, and Morten Videbæk Pedersen. *On the Need of Network coding for Mobile Clouds*. ISBN 978-963-313-033-9, Paper presented at Automation and Applied Computer Science Workshop, Budapest, Hungary. 2011.

[21] Frank Fitzek, Janus Heide, Morten Videbæk Pedersen, Gergö Ertli, and Markos Katz. "Multi-Hop versus Overlay Networks: A Realistic Comparison Based on Energy Requirements and Latency". In: *IEEE VTS Vehicular Technology Conference. Proceedings* (2011), pp. 1 –5. ISSN: 1550-2252.

[22] Peter Vingelmann, Morten Videbæk Pedersen, Frank Fitzek, and Janus Heide. "Multimedia distribution using network coding on the iphone platform". In: *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*. ACM Conference on Computer-Human Interaction, 2010. ISBN: 978-1-4503-0168-8.

[23] Frank Fitzek, Morten Videbæk Pedersen, Janus Heide, and Muriel Medard. "Network Coding Applications and Implementations on Mobile Devices". In: *Proceedings of the 5th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. ACM Conference on Computer-Human Interaction, 2010. ISBN: 978-1-4503-0278-4.

[24] Janus Heide, Morten Videbæk Pedersen, Frank H.P. Fitzek, and Torben Larsen. "Connecting the Islands - Enabling Global Connectivity through Local Cooperation". In: *The 2nd International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE)*. Berlin, Germany, 2009.

[25] Gian Paolo Perrucci, Morten Videbæk Pedersen, Tatiana Madsen, and Frank Fitzek. "Energy Evaluation for Bluetooth Link Layer Packet Selection Scheme". In: *European Wireless 2009*. Aalborg, Denmark, 2009.

[26] Morten Videbæk Pedersen, Frank H.P. Fitzek, and Torben Larsen. "Implementation and Performance Evaluation of Network Coding for Cooperative Mobile Devices". In: *IEEE International Conference on Communications (ICC 2008)*. Beijing, China, May 2008.

[27] Janus Heide, Morten Videbæk Pedersen, Frank H.P. Fitzek, Tatiana V. Kozlova, and Torben Larsen. "Know Your Neighbour: Packet Loss Correlation in IEEE 802.11b/g Multicast". In: *The 4th International Mobile Multimedia Communications Conference (MobiMedia '08)*. Oulu, Finland, 2008.

[28] Morten Videbæk Pedersen, G.P. Perrucci, Frank H.P. Fitzek, and Torben Larsen. "Energy and Link Measurements for Mobile Phones using IEEE802.11b/g". In: *The 4th International Workshop on Wireless Network Measurements (WiNMEE 2008) - in conjunction with WiOpt 2008*. Berlin, Germany, Mar. 2008.

[29] Frank H.P. Fitzek, Morten Videbæk Pedersen, and M. Katz. "A Scalable Cooperative Wireless Grid Architecture and Associated Services for Future Communications". In: *European Wireless 2007*. Paris, France, Apr. 2007.

[30] Frank H.P. Fitzek, Morten Videbæk Pedersen, and Hajo Schulz. "Innerer Antrieb – C++-Programmierung für Symbian-Smartphones". In: *ct – Computer Technique*. Ed. by Hajo Schulz. Vol. 8. www.heise.de. Heise, Mar. 2007, pp. 196–201.

Book Chapters

[31] Morten Videbæk Pedersen, Janus Heide, and Frank H.P. Fitzek. "Finite Field Arithmetics for Network Coding". In: *Network Coding: A Hands-on Approach (tentative title)*. Accepted. Wiley Books, 2013 (expected).

[32] Janus Heide, Morten Videbæk Pedersen, Frank Fitzek, and Torben Larsen. "Network Coding in the Real World". In: *Network Coding: Fundamentals and Applications*. Ed. by Muriel Medard and Alex Sprintson. 1st ed. Academic Press, 2011. ISBN: 978-0123809186.

[33] Qi Zhang, Janus Heide, Morten Videbæk Pedersen, Frank Fitzek, Jorma Lilleberg, and Kari Rikkinen. "Network Coding and User Cooperation for Streaming and Download Services in LTE Networks". In: *Network Coding: Fundamentals and Applications*. Ed. by Muriel Medard and Alex Sprintson. 1st ed. 2011; 5. Academic Press, Incorporated, 2011, pp. 115–140. ISBN: 978-0123809186.

[34] Morten Videbæk Pedersen, Janus Heide, Frank H.P. Fitzek, and Tony Torp. "Getting Started with Qt". In: *Qt for Symbian*. 1st ed. Wiley Books, 2010, pp. 13–28. ISBN: 978-0-470-75010-0.

[35] Morten Videbæk Pedersen and Frank H.P. Fitzek. "Mobile Peer To Peer Networking – The Symbian C++ Programming Environment". In: ed. by Frank H.P Fitzek and Hassan Charaf. Wiley Books, 2009. Chap. 3.

[36] Morten Videbæk Pedersen and Frank H.P. Fitzek. "Mobile Peer To Peer Networking – Introduction to Bluetooth Communication on Mobile Devices". In: ed. by Frank H.P Fitzek and Hassan Charaf. Wiley Books, 2009. Chap. 4.

[37] G.P. Perrucci, Frank H.P. Fitzek, and Morten Videbæk Pedersen. "Heterogeneous Wireless Access Networks: Architectures and Protocols – Energy Saving Aspects for Mobile Device Exploiting Heterogeneous Wireless Networks". In: ed. by Ekram Hossain. Springer, 2008. Chap. tbd.

[38] Morten Videbæk Pedersen and Frank H.P. Fitzek. "Mobile Phone Programming – Symbian/C++". In: ed. by Frank H.P. Fitzek and F. Reichert. Springer, 2007. Chap. 4, pp. 95–138.

[39] Morten Videbæk Pedersen and Frank H.P. Fitzek. "Mobile Phone Programming – The Walkie Talkie Application". In: ed. by Frank H.P. Fitzek and F. Reichert. Springer, 2007. Chap. 12, pp. 275–279.

[40] Morten Videbæk Pedersen and Frank H.P. Fitzek. "Mobile Phone Programming – SMARTEX:The SmartME Application". In: ed. by Frank H.P. Fitzek and F. Reichert. Springer, 2007. Chap. 11, pp. 271–274.

[41] Morten Videbæk Pedersen, G. Perrucci, T. Arildsen, T. Madsen, and Frank H.P. Fitzek. "Mobile Phone Programming – Cross-Layer Example for Multimedia Services over Bluetooth". In: ed. by Frank H.P. Fitzek and F. Reichert. Springer, 2007. Chap. 18, pp. 363–371.

Posters

[42] Morten Videbæk Pedersen, Janus Heide, Peter Vingelmann, Leonardo Militano, and Frank H.P. Fitzek. "Network Coding on Mobile Devices". In: *Workshop on Network Coding, Theory and Applications (NetCod)*. Lausanne, Switzerland, 2009.

Patents

[43] Janus Heide, Morten Videbæk Pedersen, Frank H.P. Fitzek, and Qi Zhang. *Intrinsic Information Conveyance in Network Coding*. Patent. Pat. 800.0406.U1 (US), pending. Mar. 19, 2010. 2012.

References

[44] Engadget. *Facebook Statistics*. http://www.engadget.com/2012/02/01/facebook-ipo-commences/. 2012.

[45] H. P. Frank and Fitzek Katz. *Cooperation in Wireless Networks: Principles and Applications: Real Egoistic Behavior Is to Cooperate!* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 140204710X.

[46] Morten Holm Larsen, Petar Popovski, and Søren Vang Andersen. "Cooperative Communication with Multiple Description Coding". In: *Cooperation in Wireless Networks*. Ed. by Frank Fitzek and Marcos Katz. IEEE Computer Society Press, 2006. ISBN: 9781402047107.

[47] Gian Paolo Perrucci, Frank Fitzek, Qi Zhang, and Marcos Katz. "Cooperative Mobile Web Browsing". In: *EURASIP Journal on Wireless Communications and Networking* 2009 (2009). ISSN: 1687-1472.

[48] Bart Giordano. "Transforming Small Mobile Devices into Full-Featured Wi-Fi Access Points". In: *White Paper Marvell Simiconductor Inc.* (2009).

[49] T.E. Hunter and A. Nosratinia. "Distributed protocols for user cooperation in multi-user wireless networks". In: *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*. Vol. 6. 2004, 3788 –3792 Vol.6. DOI: 10.1109/GLOCOM.2004.1379077.

[50] Jong-Woon Yoo and Kyu Ho Park. "A Cooperative Clustering Protocol for Energy Saving of Mobile Devices with WLAN and Bluetooth Interfaces". In: *Mobile Computing, IEEE Transactions on* 10.4 (2011), pp. 491 –504. ISSN: 1536-1233. DOI: 10.1109/TMC.2010.161.

[51] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. "Network information flow". In: *Information Theory, IEEE Transactions on* 46.4 (2000), pp. 1204 –1216. ISSN: 0018-9448. DOI: 10.1109/18.850663.

[52] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. "A random linear network coding approach to multicast". In: *IEEE TRANS. INFORM. THEORY* 52.10 (2006), pp. 4413–4430.

[53] D. Koutsonikolas, Y.C. Hu, and Chih-Chun Wang. "Pacifier: High-Throughput, Reliable Multicast without "Crying Babies" in Wireless Mesh Networks". In: *INFOCOM 2009, IEEE*. 2009, pp. 2473 –2481. DOI: 10.1109/INFCOM.2009.5062175.

[54] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. "Trading structure for randomness in wireless opportunistic routing". In: *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '07. Kyoto, Japan: ACM, 2007, pp. 169–180. ISBN: 978-1-59593-713-1. DOI: 10.1145/1282380.1282400.

[55] M. Wang and Baochun Li. "How Practical is Network Coding?" In: *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*. IEEE, June 2300, pp. 274–278. ISBN: 1-4244-0476-2. DOI: 10.1109/IWQOS.2006.250480.

[56] Randal E. Bryant and David R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. 2nd. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0136108040, 9780136108047.

[57] Poul-Henning Kamp. "You're Doing It Wrong". In: *Queue* 8.6 (), 20:20–20:27. ISSN: 1542-7730. DOI: 10.1145/1810226.1814327.

[58] Seung-Hoon Lee, Uichin Lee, Kang-Won Lee, and Mario Gerla. "Content Distribution in VANETs Using Network Coding: The Effect of Disk I/O and Processing O/H". In: *Proceedings of the Fifth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON 2008, June 16-20, 2008, Crowne Plaza, San Francisco International Airport, California, USA*. IEEE, 2008, pp. 117–125. DOI: http://dx.doi.org/10.1109/SAHCN.2008.24.

[59] Yannis Smaragdakis and Don Batory. "Mixin layers: an object-oriented implementation technique for refinements and collaboration-based designs". In: *ACM Trans. Softw. Eng. Methodol.* 11.2 (Apr. 2002), pp. 215–255. ISSN: 1049-331X. DOI: 10.1145/505145.505148.

[60] Gilad Bracha and William Cook. "Mixin-based inheritance". In: *Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*. OOPSLA/ECOOP '90. Ottawa, Canada: ACM, 1990, pp. 303–311. ISBN: 0-89791-411-2. DOI: 10.1145/97945.97982.

[61] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. "Composing high-performance memory allocators". In: *SIGPLAN Not.* 36.5 (May 2001), pp. 114–124. ISSN: 0362-1340. DOI: 10.1145/381694.378821.

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

[63] Christina Fragouli and Emina Soljanin. "Network Coding Fundamentals". In: *Foundations and Trends in Networking* Vol. 2, Issue 1 (2007), pp. 1–133. DOI: NA.

List of Abbreviations

CPU Central Processing Unit 27

D2D Device To Device 11

ECP ENOC Cooperation Protocol 10

ENOC Evolved Network COding 26

FEC Forward Error Correction 16

LTE Long Term Evolution 16

MIT Massachusetts Institute of Technology 26

MDC Multiple Description Coding 3

NOCE Network COding Evolved 26

NBC National Broadcasting Company 26

PEP Packet Erasure Probability 10

RLNC Random Linear Network Coding 6

SIMD Single Instruction Multiple Data 7

TCP Transmission Control Protocol 16

TPC Technical Program Committee 26

UDP User Datagram Protocol 16

WLAN Wireless Local Area Network 15

Contributions Included In This Thesis

Paper 1

Mobile Clouds: The New Content Distribution Platform

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

Institute of Electrical and Electronics Engineers. Proceedings, Vol. 100, 13.05.2012.

INVITED PAPER

Mobile Clouds: The New Content Distribution Platform

In this paper, the future of digital media content distribution using mobile clouds is introduced and the impact of social networks on sharing content and other limited resources such as spectrum is highlighted.

By Morten V. Pedersen, *Member IEEE*, and Frank H. P. Fitzek, *Senior Member IEEE*

ABSTRACT | This paper discusses the future of content distribution among mobile devices forming the so-called mobile clouds. This paper introduces the current-technology-based problems of the approach, but also highlights its future potential. One core element of this paper is the technical development in this area and the social paradigm that will be used to create cooperation among users. We conclude that the future of mobile clouds will be in novel technologies such as network coding as well as in combination with social networks in order to boost cooperation among users as well as connect people over the shared content.

KEYWORDS | Cooperation; mobile clouds; network coding; social networks

I. INTRODUCTION

With the dramatic evolution of mobile phones come changes to how we use these devices. From simple phone calls in the past, today the mobile phone is the main source of information storing our favorite songs and videos.

But storing the mobile content is not the final goal. Users like to share content. For this purpose, all social networks such as Facebook allow upload of content of any kind. But users want more than just a common storage of their content. They want to enjoy the content together with their friends simultaneously. This is not a new trend. In the very early days of the Walkman there were already two headset jacks. The great success of the TV as a social medium was that we could watch it together and talk about it later on.

With the introduction of the mobile phone, the consumption of content became more asynchronous. One of the reasons is that the content we watch is a downloaded content. Live streaming such as in Internet protocol television (IPTV) is not widely deployed yet. Unfortunately, our mobile phones and our networks are not completely ready yet to support the described usage scenario. Current networks, for example, have difficulties to support multicast or broadcast services for mass events such as rock concerts or sport events, especially if the receivers are spatially correlated. Furthermore, the social interaction over such content cannot be globalized, but is limited to people which are spatially or socially close to each other. Therefore, the trend tends to share the content in a more cooperative way from device to device (D2D).

Without going into detail, sharing is not limited to content but follows a more general concept of sharing resources. Sharable resources are, e.g., spectrum, computational power, apps, onboard sensors, achieved knowledge as well as the aforementioned content [1], [2]. After answering the question on what will be shared in the mobile clouds, the follow-up question is how will the sharing be realized? Here we will highlight two aspects, namely, the technology side and the social side. The technology side looks into the efficient sharing of the content in a mobile cloud, while the social side discusses different forms of cooperation within the mobile cloud from forced cooperation, altruism, and new forms of cooperation.

II. MOBILE CLOUDS: CURRENT PROBLEMS

Since the very early days of mobile communication, cellular and centralized concepts have dominated the

Manuscript received February 17, 2012; accepted February 21, 2012. Date of publication April 4, 2012; date of current version May 10, 2012. This work was supported in part by the Danish Ministry of Science, Technology and Innovation under the CONE project (Grant 09-066549/FTP). This work was also supported by the Danish Council for Independent Research (Sapere Aude program) under Green Mobile Clouds Project 10-081621/FTP.

The authors are with the Department of Electronic Systems, Aalborg University, Aalborg DK-9220, Denmark (e-mail: mvp@es.aau.dk; ff@es.aau.dk).

Digital Object Identifier: 10.1109/JPROC.2012.2189806

communication world. This old paradigm limits us. We should break with these concepts and start to think about D2D concepts. As the content is not necessarily stored in the overlay network, devices might convey information directly to the neighboring devices without any help of the overlay network. This technical solution maps perfectly with the social need to share with people who are close to us.

The problem is that the mobile platforms are not, or even worse not anymore, ready for this. Such an approach could be realized by the globally accepted WiFi technology. Most mobile phones, whether featured phones or smartphones, are equipped with WiFi already. In order to not be dependent on any overlay network, *ad hoc* WiFi would be the best choice. But as the first WiFi-enabled phones were still able to support *ad hoc* WiFi, some of the newer devices do not allow it anymore. More precisely, the devices are allowed to join an *ad hoc* network but cannot establish such a network. Even if the *ad hoc* capabilities are supported as, for example, on some Android phones or the Nokia N9, the performance of *ad hoc* communication was reported to be rather low. Fortunately, WiFi direct is now implemented on the newest phones. This will boost the D2D communication platform. Whatever technology is available, network coding has to be implemented for the efficient exchange of the content. In case the content is not stored on any participating mobile, but within the Internet, D2D still offers many benefits. Without the D2D capability of mobile devices in close proximity, the network operator needs to make sure that each participating mobile phone gets full information. In case of D2D, the network would just pump enough information into the D2D group and leave it up to the phones to exchange the missing parts. Such an approach leads to not only energy saving for the mobile phones, but also energy and bandwidth savings for the network operators.

Another problem has to do with legal aspects: Which content can be used for D2D networks? Downloadable content is most often digital right management (DRM) protected. This last problem may be solved by identifying users that would like to share the same content and setting up an efficient transmission among them.

III. MOBILE CLOUDS: THE FUTURE

Here we highlight the future of the mobile clouds with respect to content distribution. We will examine two main aspects, namely, the technological and social domains.

A. Technological Domain

Currently, there is a lot of research work going on to make the communication within the mobile clouds feasible and highly attractive for users, network operators, and service providers. A key element here is the network coding. Introduced by Ahlswede *et al.* [3], the main contribution was done later by Ho *et al.*, who introduced the random linear network coding [4]. Using network coding for the distribution of content within a mobile cloud leads to energy savings, bandwidth savings, delay reduction, privacy assurance, as well as preventing false packet injection. It has been shown that the implementation of network coding on any mobile platform is feasible and the energy spent for operating network coding is less than the energy savings that will be achieved by the reduced bandwidth requirements. First applications for mobile phones have been prototyped [5], showing the benefits of network coding. Conceivably mobile clouds will be powered in the future by network coding due to the list of benefits given beforehand. Fig. 1 shows one of the first content sharing demonstrations [5]. One sender shares a video with 16 receivers in close proximity.

B. Social Domain

A more interesting question is how can cooperation among mobile phones be achieved? It is critical to understand the reasoning behind users' cooperation or defection in order to influence users' willingness to participate. In Fig. 2, four different modes of cooperation are shown: forced cooperation, technology-enabled cooperation, socially enabled cooperation, and altruism. Forced cooperation takes place if, for example, the content is shared to any requesting device without asking the content holder whether she or he agrees. As this may be seen as disadvantageous for the content holder, it has huge benefits for the network operator. If mobile devices have different owners, cooperation becomes more difficult. In its easiest form, cooperation takes place if it is based on altruism. Here, the content holder is willing to share content with friends and family and even strangers. As shown by Hamilton in 1963 [6], in human science, some mobile devices willingly sacrifice some of their own

Fig. 1. *Example of sharing a movie locally among mobile devices using WiFi technology for the iOS platform.*

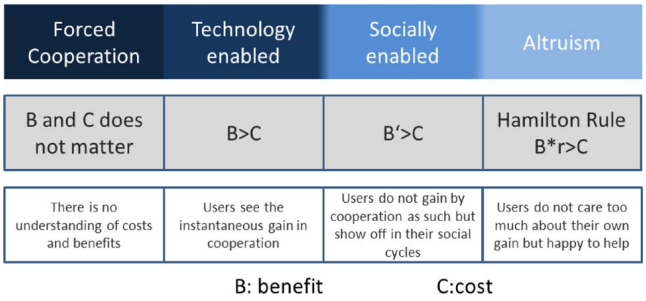


Fig. 2. ***Different cooperation modes.***

benefits in favor of others, as long as $B^*r > C$, where B is the expected benefit for the receiver of the donation, C is the cost involved for the donor, and r is the relationship between the two entities.

The easiest way to encourage the use of cooperative technologies is to create situations where the instant benefit (B) is larger than the cost (C) of undertaking cooperation for all participating users. The better we design the technology the lower the costs are. Cooperative IPTV, as introduced in [7], where users cooperate during downloading of the prefetched TV content, undoubtedly exemplifies this point. In order to find more scenarios, new communication protocols and techniques are needed. Should the benefit of cooperation remain unclear to all users, social reinforcement will kick in. In such a scenario, we predict one or more mobile devices gaining from the cooperation (we call them the *receivers*) and one or more entities who invest in cooperation but do not gain (we call them the *investors*). While the gain for the *receivers* is clear, the gain for the *investors* is not. It is well documented that most users of mobile devices are members of social networks such as Facebook and Google+. When *investors* help establish cooperation with little to no perceived benefit to them, their efforts should be rewarded in social networks with a different kind of benefit (B) (see Fig. 3). This can be done by simple notification or other gamification concepts. The gain for the *investors* is therefore within the social domain where they will obtain rewards from the *receivers* as well as their own social graph. While forced cooperation and altruism are two well-known concepts that represent the state of the art, technology and socially enabled mobile clouds are undoubtedly beyond the state of the art. In contrast to any other tit-for-tat cooperation scheme such as FON [8] or BitTorrent [9], the cooperative exchange is not repaid with equal currency but is rewarded within a new dimension: the social domain.

Currently, social networks dominate the information and communication technology (ICT) world and will become even more pervasive in the future. Social networks will not only boost cooperation among mobile devices, but also they will allow social commentary about the content currently or recently shared. Such an approach has several advantages for the:

- *User*: content sharing is faster and more energy efficient compared to cellular download; simultaneous consuming of digital content (music, video, pictures);
- *network operator*: local sharing will offload the overlay networks where spectrum is a scarce resource;
- *service/content provider*: content will be spread quickly and viral loops will be established, spreading interesting content with larger speed.

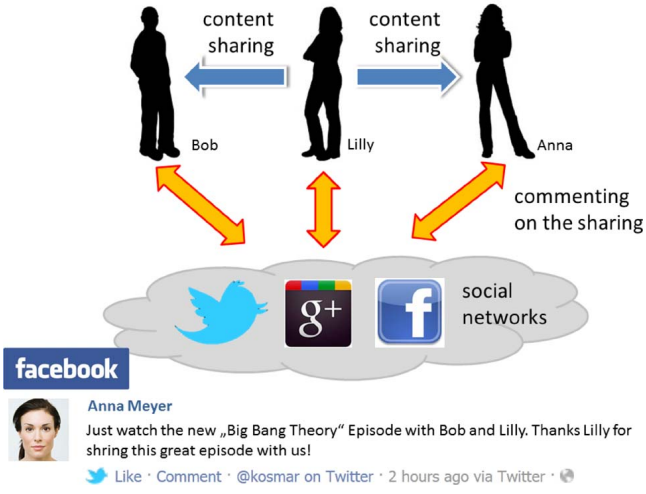


Fig. 3. ***Local sharing with social networks to enable cooperation and discuss the digital content.***

IV. CONCLUSION

In this paper, the future of content distribution for digital media using mobile clouds is introduced. The mobile cloud concept foresees that mobile devices connect to each other directly without any help of the overlay network. For the actual sharing among mobile devices new technologies such as network coding are the key enabler for support of energy saving, privacy, security, data protection, and fast exchange of data. This new architecture fits the needs of users who would like to enjoy the digital content together. In order to boost cooperation, especially for users who do not know one another, the social networks are introduced. By means of social networks, mismatch in cooperation gain can be balanced out.

In this paper, we only highlighted the impact of social networks on content sharing only, but in the future, users might also share other resources such as spectrum, onboard sensor information with each other, using the reporting capabilities of the social networks. ■

REFERENCES

[1] F. H. P. Fitzek and M. Katz, *Cognitive Wireless Networks: Concepts, Methodologies and Visions Inspiring the Age of Enlightenment of Wireless Communications*. New York: Springer-Verlag, Jul. 2007, ISBN: 978-1-4020-5978-0.

[2] F. H. P. Fitzek and M. Katz, Eds., *Cooperation in Wireless Networks: Principles and Applications—Real Egoistic Behavior Is to Cooperate!*. New York: Springer-Verlag, Apr. 2006, ISBN: 1-4020-4710-X.

[3] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[4] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proc. IEEE Int. Symp. Inf. Theory*, 2003, DOI: 10.1109/ISIT.2003.1228459.

[5] P. Vingelmann, F. H. P. Fitzek, M. V. Pedersen, J. Heide, and H. Charaf, "Synchronized multimedia streaming on the iPhone platform with network coding," *IEEE Commun. Mag.*, ser. Consumer Communications and Networking, vol. 49, no. 6, pp. 126–132, Jun. 2011.

[6] W. D. Hamilton, "The evolution of altruistic behavior," *Amer. Naturalist*, vol. 97, pp. 354–356, 1963.

[7] Q. Zhang, J. Heide, M. V. Pedersen, and F. H. P. Fitzek, "MBMS with user cooperation with network coding," in *Proc. IEEE Global Telecommun. Conf.*, Houston, TX, 2011, DOI: 10.1109/GLOCOM.2011.6133643.

[8] FON. [Online]. Available: http://corp.fon.com

[9] BitTorrent. [Online]. Available: http://www.bittorrent.com/

ABOUT THE AUTHORS

Morten V. Pedersen (Member, IEEE) received the B.Sc. degree in electronics engineering and the M.Sc. degree in wireless communication from University of Aalborg, Aalborg, Denmark, in 2007 and 2009, respectively, where he is currently working towards the Ph.D. degree in the Department of Electronic Systems.

Since January 2006, he has been working in the Mobile Device Research Group at Aalborg University, where his primary focus has been implementation and performance evaluation of cooperative networking protocols and methods. As a member of the Mobile Devices team, he has been responsible for teaching activities on the smartphone platforms since 2006. He has coauthored and published several peer-reviewed journal and conference papers, and multiple book chapters. His main research interests are mobile programming, cooperative communication, network coding, and network performance evaluation.

Mr. Pedersen was appointed Nokia Champion in 2010. He served as a local organizer of the 2009 European Wireless Conference. He has been involved in the preparation and organization of the Mobile Developer Days 2007 and 2008, a developer conference focusing on mobile devices with approximately 100 participants.

Frank H. P. Fitzek (Senior Member, IEEE) received the Diploma (Dipl.-Ing.) degree in electrical engineering from the University of Technology—Rheinisch-Westfälische Technische Hochschule (RWTH), Aachen, Germany, in 1997 and the Ph.D. (Dr.-Ing.) degree in electrical engineering from the Technical University Berlin, Berlin, Germany, in 2002.

In 2002, he became an Adjunct Professor at the University of Ferrara, Ferrara, Italy. Currently, he is a Professor in the Department of Electronic Systems, University of Aalborg, Aalborg, Denmark, heading the Mobile Device group. He cofounded the startup company acticom GmbH in Berlin, Germany, in 1999. He has visited various research institutes including the Massachusetts Institute of Technology (MIT, Cambridge), VTT Technical Research Centre of Finland, and Arizona State University, Tempe. His current research interests are in the areas of wireless and mobile communication networks, mobile phone programming, network coding, cross layer as well as energy-efficient protocol design and cooperative networking.

Dr. Fitzek received the 2005 YRP award for the work on MIMO MDC and the Young Elite Researcher Award of Denmark. He received the Nokia Champion Award several times in a row from 2007 to 2011. In 2008, he was awarded the Nokia Achievement Award for his work on cooperative networks. In 2011, he received the SAPERE AUDE research grant from the Danish Government.

Paper 2

On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices

Péter Vingelmann and Morten Videbæk Pedersen, Frank H. P. Fitzek and Janus Heide

Globecom. IEEE Conference and Exhibition, 05.12.2011.

On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices

Péter Vingelmann*†, Morten Videbæk Pedersen†, Frank H. P. Fitzek†, and Janus Heide†
*Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary
†Department of Electronic Systems, Faculty of Engineering and Science, Aalborg University, Denmark

***Abstract*—This paper investigates the possibility of packet error recovery in a cooperative cluster of mobile devices. We assume that these devices receive data from a broadcast transmission on their primary network interface (e.g. LTE network), and they are using a secondary network interface (e.g. ad hoc WLAN network) to form a cooperative cluster in order to exchange missing data packets among each other. Our goal is to devise a protocol that minimizes the number of packets exchanged on the secondary network whilst maximizes the number of packet errors recovered on the primary network. Moreover, we aim to repair the packet losses on-the-fly (as the data is being received), which also imposes real-time constraints on the protocol. We propose a solution based on random linear network coding to form cooperative clusters of mobile devices to facilitate the efficient exchange of information among them. We also introduce a demo application that implements this technique on Nokia phones. Then we present our testbed and the collected measurement results in order to evaluate the performance of our protocol.**

I. Introduction

Network coding has received a lot of attention lately [1], [2], [10], [8], [12]. Researchers have shown that network coding has its clear advantages, especially for wireless and mobile multi-hop communication systems. Our prior work has been focused on the feasibility and tuning of network coding for mobile platforms [5], [14], [13]. Later those findings were confirmed by other researchers [15], so we could conclude that network coding is feasible on embedded devices, and its benefits in terms of energy consumption and bandwidth usage are realistic. Researchers in [11], [4] proposed solutions that utilize network coding in cooperative clusters in various communication scenarios.

The work in this paper is also focusing on protocol design based on network coding. In particular, we investigate the possibility of using cooperative clusters of mobile devices for packet error recovery in content distribution systems. The main architecture of these systems has not changed much in the past decade. In the mobile world, we still use a highly centralized client/server architecture, where the overlay network is providing the content, and the mobile users are merely consuming it.

The protocol proposed here takes advantage of the mobile devices themselves by forming cooperative clusters with the help of network coding. We attempt to enhance the traditional client/server communication pattern by allowing neighboring devices to communicate directly to make the content distribution more efficient.

This paper is organized as follows. Section II presents our scenario and gives an overview on network coding. Section III discusses the features of our protocol and introduces our application. Section IV presents measurement results. Future works are discussed in Section V, and the final conclusion is drawn in Section VI.

II. Scenario

In our scenario a source wants to reliably transmit the same media file or media stream to several receivers. We assume that these receivers are connected to the source via their primary network interface (e.g. 4G or LTE network). In state-of-the-art systems the source *broadcasts* the data on this interface, and it also applies some sort of coding scheme (e.g. Raptor coding) to fix the packet erasures on the receivers. Since packet losses are quite common in wireless networks, this coding may require a large overhead. Therefore we assume that the receivers are also able to communicate with each other using their secondary network interface (e.g. ad hoc WLAN). Thereby cooperative clusters can be formed of several receivers to exchange missing data packets among each other. This basic scenario is depicted in Figure 1.

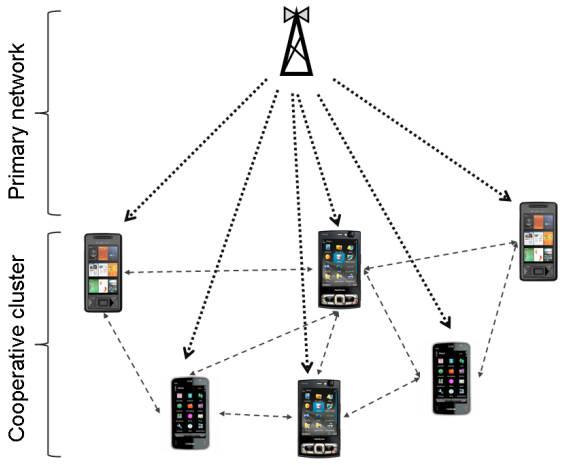


Fig. 1. A source transmitting data to multiple receivers.

The fundamental problem with broadcast packets is the frequent losses of packets in real-life wireless networks [6]. Since channel conditions are not ideal, we cannot expect that all

broadcast packets are delivered to all receivers. Packet losses must be corrected by using some sort of retransmission scheme to ensure reliability. As a naive solution, the individual nodes can request all their missing packets from the original source. The end result would be similar to the Automatic Repeat-reQuest (ARQ) mechanism used in one-to-one transmission protocols, since every lost packet would be transmitted again. This strategy is sub-optimal if packet losses are uncorrelated, as each retransmission is only useful to those receivers that have lost the given packet in the first place. It is likely that a single retransmission will only benefit a single receiver.

The impact of each retransmission can be maximized by using network coding [1], [2], [9], [5]. Researchers have shown that network coding can provide several advantages, namely improved throughput, robustness, security and lower complexity in communication networks [7], [3].

A. Network Coding

The basic operations performed in a network coding system are depicted in Figure 2. To lower the computational complexity of coding operations, large files or continuous streams are typically split into several equal sized chunks, also called *generations* [2], each consisting of g packets.

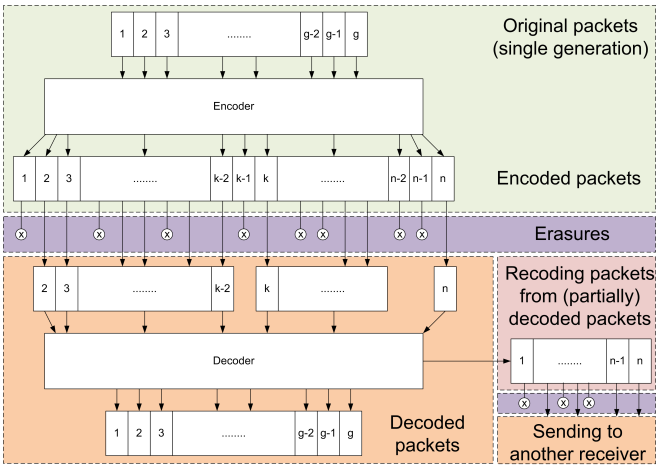


Fig. 2. Overview of Network Coding

The *encoder* (the top component in Figure 2) generates and transmits linear combinations of the original data packets in the current generation. Addition and multiplication are performed over a Galois field, therefore a linear combination of several packets will result in a packet having the same size as one of the original packets. With Random Linear Network Coding (RLNC), the coding coefficients are selected at random. Note that any number of encoded packets can be generated for a given generation. The middle layer represents the *wireless channel*, where packets are lost depending on the channel conditions. The received encoded packets are passed to the *decoder* (the bottom component in the figure), which will be able to reconstruct the original data packets after receiving at least g linearly independent packets.

The receiver nodes are also allowed to generate and send new encoded packets, even before decoding the entire generation. They form new linear combinations of the packets that they have previously received. This operation is known as *recoding*, and it is a unique feature of network coding. Traditional coding schemes require the original data to be fully decoded before it can be encoded again.

Another advantage of network coding is that it makes *"perfect coordination"* possible, where an arbitrary number of sending nodes can be used to serve the same generation to a receiver. Moreover, a receiver is no longer required to gather all data packets one-by-one, it can simply "hold a bucket" for a generation until it is full, that is enough linearly independent encoded packets are received. With RLNC, the randomly generated coding coefficient vectors from different senders are linearly independent with high probability (depending on which Galois field is used). Consequently, there is only a minimal need for signaling among the cooperating nodes.

III. Protocol Design

Our goal is to devise a protocol that minimizes the number of packets exchanged among the mobile clients, whilst maximizes the number of packet errors recovered. Moreover, we aim to repair the packet losses on-the-fly (as the data is being received), which imposes real-time constraints on the protocol. We call our protocol ECP (ENOC Cooperation Protocol).

We assume that the primary network is an LTE network, which uses systematic Raptor coding for broadcast transmissions. Thus the raw symbols (uncoded data packets) are transmitted first, and they are followed by several encoded packets to repair the losses. We intend to devise a best-effort protocol that uses the secondary network established among the receivers to conceal a significant part of these losses from the LTE Raptor decoder. It is important to note that ECP is not intended to provide full reliability, which remains the responsibility of the Raptor decoder. We assume that the protocol has read access to the raw symbols received on the primary network, and it can also write back newly recovered symbols to the Raptor decoder buffer, thereby significantly lowering the perceived packet error rate and the required overhead.

A. Protocol operation

ECP is based on the principles of network coding, since the receivers cooperate by sending encoded or recoded messages, which contain information that is most likely innovative for all peers. The basic unit of operation is a generation, meaning that the cooperative cluster is only trying to fix a specific generation at any given time. The network nodes may form a new cluster for the next generation.

After receiving a certain generation on the primary network, a receiver can broadcast a NACK (Negative Acknowledgment) message on the secondary network if it has experienced any packet losses. This is a retroactive trigger mechanism in ECP. We apply semi-random back-off intervals to prevent multiple nodes from broadcasting NACK messages at the same time.

The back-offs are chosen so that it is more likely that the worst receiver sends out the first NACK. These messages contain information about how many packets were lost on the receiver.

When the other devices within range receive this packet, they will suppress their own NACKs, and in response they schedule several encoded data packets to be sent at a specific speed. The devices generate and broadcast encoded packets, which also convey information about their own packet losses.

Since RLNC is used to achieve perfect coordination, encoded/recoded packets from any of the nodes can be equally useful. Senders do not have to pay attention to select specific packets for specific receivers.

Consequently, the most essential question here is how many packets the devices should schedule and transmit in response to a NACK message. They can simply broadcast as many as the worst receiver needs. The nodes constantly gain information about the others' knowledge with every encoded packet they receive. These updates can be used to continuously adjust the remaining number of packets to be sent.

This simple approach however leads to sub-optimal performance in most cases. If there are 3 or more cooperating devices, then the task of "filling up" the worst receiver should be equally divided among the others. For example, suppose that there are 4 devices and the worst one needs 15 packets, then the other 3 can send 5 packets each.

If we have only 2 cooperating nodes, then it is very likely that they have some common erasures. Consequently, full recovery is often not possible in this case, and the devices should send less packets than the other one has lost. Specific information about the individual packet losses is necessary to determine the combined knowledge of the cooperative cluster. We include a short bit vector (called the knowledge vector) in all protocol messages that explicitly indicates which packets of the current generation are currently available on the sender of the message. The network nodes can quickly calculate the combined knowledge of the cluster by bitwise OR-ing these vectors from all receivers. To determine how many innovative packets can be sent to a given node, we can take the difference of the combined knowledge and the node's latest knowledge vector.

This improved approach can yield near optimal performance under ideal channel conditions and slow transfer rates. However, as we increase the transfer rate, we observe that information about the other nodes can quickly become out-of-date thus inaccurate. This may lead to the premature termination of the repair session, i.e. the nodes may schedule less packets than necessary. Therefore we allow the receivers to send secondary NACK messages for the current generation when the others have stopped sending, but the node is still unable to decode some of the received messages.

B. Implementation

Based on the protocol design ideas outlined above, we have implemented a prototype application using the Qt framework so that we can test our protocol on any Nokia phone, desktop computer or laptop.

This demo application emulates an incoming LTE packet flow on all devices synchronously. When the application starts, it enumerates its network interfaces in order to determine its IP and broadcast addresses corresponding to the WLAN network interface. One device is used to start the simulation by broadcasting a command packet that contains all the simulation parameters. The other devices receive this packet, extract the parameters, and initiate the packet flow simulation. They all use the same random seed to generate random data packets deterministically, as if these were received on the LTE network. Packets are generated continuously with the same data rate on all clients. The devices also drop certain packets at random to simulate packet losses. Note that they do this independently from each other. Thereby all devices possess some parts of the incoming datastream, and they also have some "holes" that ECP should be able to fill on-the-fly.

We consider a single Raptor code source block, e.g. 1024 packets in a simulation. The entire block can be segmented into smaller generations (e.g. 64 packets) that are sufficiently small for network coding calculations on resource-constrained mobile devices. ECP works on these generations as depicted in Figure 3.

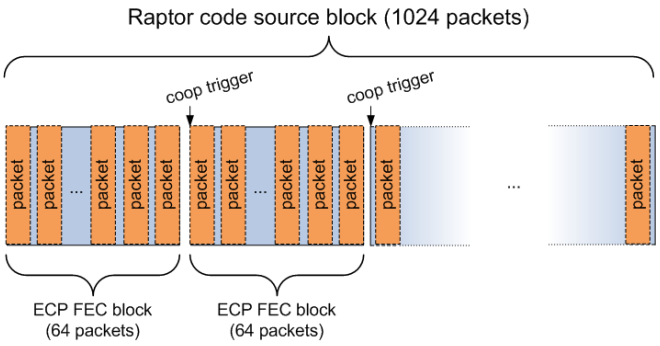


Fig. 3. Mapping between Raptor code source blocks and ECP generations.

When the first 64 packets of the emulated packet flow are (partially) received, the ECP trigger mechanism is activated, and the nodes generate a random back-off interval before sending a NACK (Negative Acknowledgment) message. When they receive a NACK, the devices generate and broadcast several encoded packets with a specified data rate in order to repair the packet losses in the cluster.

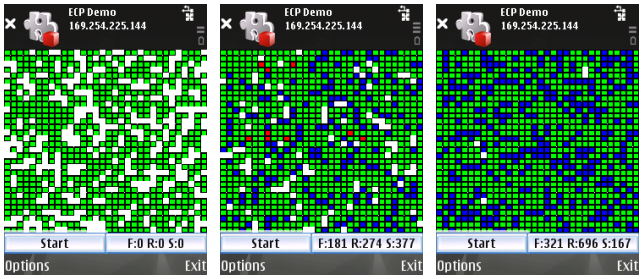
Network coding involves a computational overhead which might be prohibitive from a practical point of view. The authors in [5] proposed an efficient solution for mobile devices, namely to use the binary Galois field, GF(2) to simplify all calculations. Since this approach increases the probability of generating linearly dependent (i.e. useless) encoded packets, we have also implemented arithmetics over another finite field, GF(2^8), which is more computationally intensive, but almost totally free of linearly dependent packets. For a given simulation, all devices use either GF(2) or GF(2^8), as dictated by the source node in the simulation parameters.

The application can run simulations with different data rates, which is quite useful when we examine the implications of time constraints on the protocol performance. Moreover, data rates can be increased automatically for stress tests, and the collected simulation results from all participating devices can be transmitted to a logging server.

C. Visualization

The application also has a simple visualization grid with colors, hence we can observe as the packet losses are being repaired by the neighboring devices. In Figure 4 we show screenshots of this visualization grid after finishing simulations with 1, 2 and 6 devices.

A green box signifies a packet that was received on the simulated primary network. A red box signifies an encoded packet that was received from another device, but it has not been decoded yet, thus it can be considered ”dirty data”. A blue box signifies a packet that was originally lost on the primary network, but it has been recovered by ECP. Obviously, our objective is to maximize the number of blue boxes and to eliminate all red boxes.



(a) Single device (b) One of 2 devices (c) One of 6 devices

Fig. 4. Visualization grid after complete simulations with 1024 packets and 30% simulated packet loss.

IV. Measurements

We have assembled a testbed to demonstrate the capabilities of our protocol, and to refine its design. The cooperative cluster consists of 6 Nokia N95 (or N95 8GB version) mobile phones that run the demo application described above. These devices form an ad hoc WLAN network for short-range communication. All ECP packets are transmitted on this network.

It is important to design ECP so that it can work well with a various number of neighboring devices. In this testbed we can test its operation with 1-6 devices. With one device, the objective is to minimize outgoing traffic, although we have to make sure that the devices can detect each other’s presence and switch on cooperation when necessary. With 6 devices, we also aim to send as few packets as possible to minimize the overall energy consumption.

A. Performance evaluation

The main purpose of this testbed is to evaluate the performance of ECP on actual mobile devices, since it is a vital step in protocol design to gain feedback from real networks.

Our first performance metric is the required overhead to complete the transmission, as measured on the primary source node (i.e. the LTE base station).

In Figure 4 we can easily count the white ”holes”, each of those would require a retransmission from the source. In these simulations we used a source block consisting of 1024 packets with 30% simulated packet loss. We expect around 300 lost packets if a device is alone, that is why we see around 300 holes in the left screenshot (4a) after a finished simulation.

The middle screenshot (4b) illustrates the cooperation gain with just 2 devices. In this case we were able to fill 70% of the holes, that means a 70% reduction in the required overhead. Some packets were not fully decoded here (red boxes), which indicates sub-optimal performance with 2 devices.

The right screenshot (4c) shows the end result of a simulation with 6 devices, where 98% of the holes were filled. Therefore only a minimal amount overhead is required from the base station in this case. There are no red boxes here, and the remaining holes are actually correlated erasures, i.e. packets that were lost on all 6 devices of the cluster.

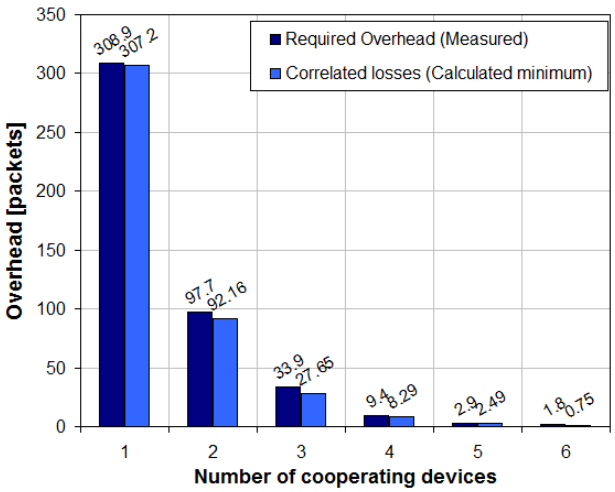


Fig. 5. Required overhead on the primary network.

Figure 5 shows the average required overhead as the number of cooperating devices increases. This graph is based on several hundred measurements performed with GF(2). As we can observe, the cooperation gain is quite significant even with 2 devices. Note that packets, that were lost on all devices, cannot be recovered. Their number is shown as correlated losses in Figure 5. In most cases ECP is able to achieve very close to the maximum cooperation gain, i.e. it can repair almost all packet losses that can be repaired by the cluster.

Our second performance metric is the number of packets exchanged on the WLAN network. The overall energy consumption of the system is heavily influenced by this measure. In the testbed, we can measure the number of transmissions (packets sent and received) on a single device, and we can aggregate these values from all devices for a given simulation.

In our previous work [17] the theoretical upper and lower bounds for this measure were established considering an ideal system based on network coding. The upper bound can be calculated by doubling the expected number of recoverable (i.e. non-correlated) packet losses on a single device. In the worst case, the device has to both send and receive that amount of packets. As we have mentioned before, the sending part can be optimized if there are more than two devices. We can divide the number of packets to be sent among all nodes. This principle is used to establish a lower bound, which is calculated for the worst receiver (on the primary network). For this device, the expected number of recoverable packet losses is higher than the average, and this is the amount of packets that it has to receive. It should also send its fair share of the expected number of recoverable losses on the second worst receiver. The lower bound is obtained by adding the number of packets received and sent. This is based on the observation that any packet exchanged on the WLAN network is either sent or received by a given node.

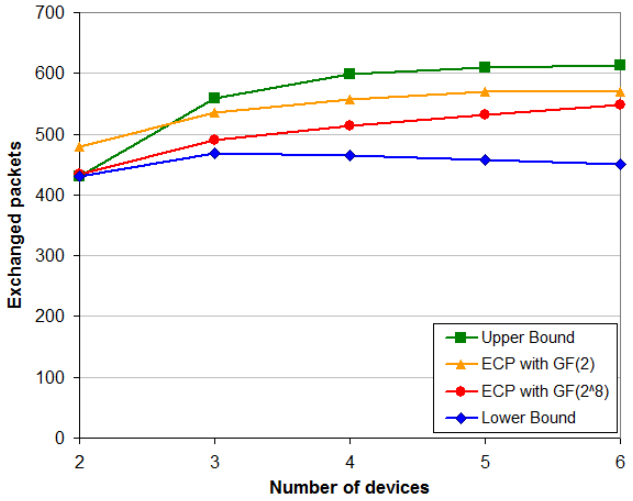
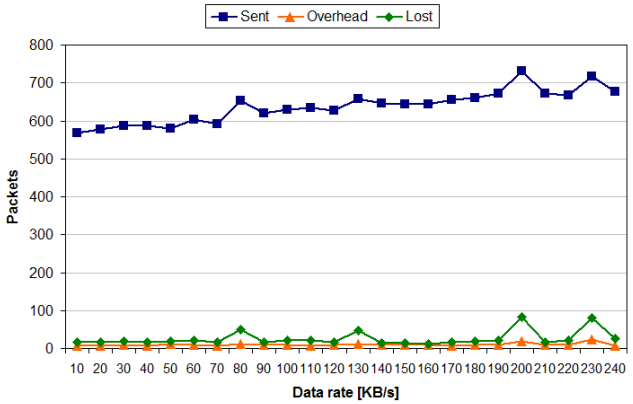


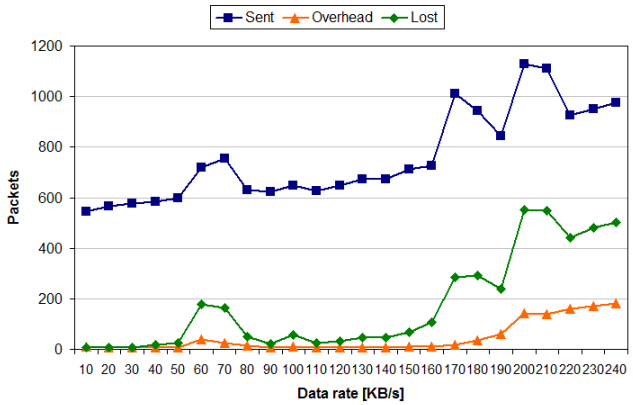
Fig. 6. Number of exchanged packets on the WLAN network as a function of the cluster size

The lower and upper bounds are plotted in Figure 6 together with the actual numbers measured on the testbed using GF(2) and GF(2^8). The performance with GF(2^8) is quite close to the lower bound for 2 and 3 devices, but it is getting worse for larger cluster sizes. On the other hand, using GF(2) results in generating some linearly dependent (i.e. useless) packets, which account for the performance gap between the two Galois fields. For 6 devices, GF(2) performs similarly to GF(2^8), which is an interesting result. It can be explained by the fact that the processing delay is significantly smaller for GF(2), therefore the protocol decisions (how many packets to send) are based on more up-to-date information.

The third performance metric is the compliance with real-time constraints. This can be very important for video playback



(a) ECP with GF(2)



(b) ECP with GF(2^8)

Fig. 7. Performance at different data rates with 4 devices

or similar services, where the packet losses should be repaired on-the-fly, i.e. faster than the native playout data rate of the video stream. In the testbed, we can adjust the data rate of the simulated LTE packet flow, and we can measure how the increased data rates influence the protocol performance. In Figure 7 we have plotted the number of total packets sent, the required overhead and the number of lost packets as measured in consecutive simulations with 4 devices with increasing data rates ranging from 10 to 240 KB/s. As we can see in Figure 7a, these performance indicators were almost unaffected when we used GF(2). We can only notice a slight increase in the number of sent packets. However, if we look at Figure 7b, we observe that the situation is much worse for GF(2^8). Encoding and decoding operations over GF(2^8) impose a heavy load on the CPU, which leads to frequent packet losses on the WLAN network. Consequently, the devices will try to send more packets to counteract these losses, and the number of sent packets increases almost twofold. As a result, the protocol is no longer able to recover all the original losses with data rates higher than 180 KB/s. For higher data rates, GF(2) performs better than GF(2^8) due to the high computational overhead associated with GF(2^8). Note that the highest data rate here,

240 KB/s corresponds to around 2 Mbit/s, which is more than sufficient for full quality videos on state-of-the-art mobile devices.

V. Future works

Before this protocol can be deployed in real-life mobile networks, we need to investigate the effects of peer mobility and dynamic behavior. Mobile devices may join and leave the cluster at any time, and typically they are also moving around, therefore static cooperative clusters are not to be expected in reality. Extensive measurements are needed to analyze the protocol performance in dynamic settings.

ECP was designed so that it can work well with multiple disjoint clusters if certain nodes move from one to another. Basically a new cluster is formed for every generation (e.g. 64 packets). If a cluster stays together for that short period of time, near-optimal performance can be achieved, and the next generation is handled by a new (possibly different) cluster.

The protocol can also be used for coverage extension in multi-hop wireless networks to provide services on mobile phones that are outside the range of 3G, 4G or LTE networks. If some nodes are *not* directly reachable by the source, then it is still possible to deliver the data stream to these receivers with the help of relaying nodes that propagate the received data farther away from the source. As it was shown in [16], the fundamental problem of data dissemination in multi-hop networks is the dynamic selection of relays and finding the proper scheduling scheme for the packet flow.

VI. Conclusion

In this paper we have introduced a protocol to facilitate the dissemination of multimedia content using cooperative clusters of mobile devices. We considered a scenario where these devices receive data from a broadcast transmission on a primary network (e.g. LTE network), and they use a secondary network (e.g. ad-hoc WLAN network) to recover packets that are lost during the transmission. We proposed a solution based on random linear network coding to facilitate the efficient exchange of information in the cooperative cluster. A demo application running on Nokia phones has been presented to show the feasibility of this approach. We observed that in most cases the protocol is able to realize almost the maximum cooperation gain, that is to recover all packets which are lost on the primary network. Meanwhile, the number of packets exchanged on the secondary network was kept close to the minimum. Moreover, we showed that packet errors can be recovered on-the-fly with data rates as high as 2 Mbit/s.

Acknowledgments

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

References

[1] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, Jul 2000.

[2] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.

[3] Christina Fragouli and Emina Soljanin. *Network Coding Applications*. Now Publishers Inc, January 2008.

[4] Z.J. Haas and T.C. Chen. Cluster-based cooperative communication with network coding in wireless networks. In *The 2010 Military Communications Conference - Unclassified Program*, pages 596–603, San Jose, California, USA, October 2010.

[5] J. Heide, M. Pedersen, F.H.P. Fitzek, and T. Larsen. Network coding for mobile devices - systematic binary random rateless codes. In *Workshop on Cooperative Mobile Networks 2009 - ICC09*. IEEE, June 2009.

[6] J. Heide, M. Pedersen, F.H.P. Fitzek, T. Madsen, and T. Larsen. Know Your Neighbour: Packet Loss Correlation in IEEE 802.11b/g Multicast. In *4th International Mobile Multimedia Communications Conference (MobiMedia 2008)*, Oulu, Finland, July 2008. ICTS/ACM.

[7] Tracey Ho and Desmond Lun. *Network Coding: An Introduction*. Cambridge University Press, 2008.

[8] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, 2006.

[9] D.E. Lucani, F.H.P. Fitzek, M. Medard, and M. Stojanovic. Network coding for data dissemination: It is not what you know, but what your neighbors know. In *RAWNET/WNC3 2009*, June 2009.

[10] Ryutaroh Matsumoto. Construction algorithm for network error-correcting codes attaining the singleton bound. *VOL.E 90-A*, 9:1729, 2007.

[11] L. Militano, F.H.P. Fitzek, A. Iera, and A. Molinaro. A genetic algorithm for source election in cooperative clusters implementing network coding. In *IEEE International Conference on Communications (ICC 2010) - CoCoNet Workshop*, May 2010.

[12] Joon-Sang Park, M. Gerla, D.S. Lun, Yunjung Yi, and M. Medard. Codecast: a network-coding-based ad hoc multicast protocol. *Wireless Communications, IEEE*, 13(5):76–81, October 2006.

[13] M.V. Pedersen, F.H.P. Fitzek, and Torben Larsen. Implementation and performance evaluation of network coding for cooperative mobile devices. *Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on*, pages 91–96, May 2008.

[14] M.V. Pedersen, J. Heide, F.H.P.Fitzek, and T. Larsen. Pictureviewer - a mobile application using network coding. In *European Wireless 2009*, Aalborg, Denmark, May 2009.

[15] H. Shojania and B. Li. Random network coding on the iphone: Fact or fiction? In *ACM NOSSDAV 2009*, June 2009.

[16] P. Vingelmann, F.H.P. Fitzek, and D. E. Lucani. Application-level data dissemination in multi-hop wireless networks. In *IEEE International Conference on Communications (ICC 2010) - CoCoNet Workshop*, May 2010.

[17] Q. Zhang, J. Heide, M. Pedersen, F.H.P. Fitzek, Jorma Lilleberg, and Kari Rikkinen. Network Coding and User Cooperation for Streaming and Download Services in LTE Networks. In *Network Coding: Fundamentals and Applications*. Academic Press, 2011.

Paper 3

Kodo: An Open and Research Oriented Network Coding Library

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

NETWORKING 2011 Workshops: International IFIP TC 6 Workshops, PE-CRN, NC-Pro, WCNS, and SUNSET 2011, Held at NETWORKING 2011, Valencia, Spain, May 13, 2011, Revised Selected Papers. Vol. 6827 Springer, 2011. p. 145-153 (Lecture Notes in Computer Science).

Kodo: An Open and Research Oriented Network Coding Library

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

Aalborg University, Denmark,
mvp@es.aau.dk,
home page: http://cone-ftp.dk

Abstract. This paper introduces the Kodo network coding library. Kodo is an open source C++ library intended to be used in practical studies of network coding algorithms. The target users for the library are researchers working with or interested in network coding. To provide a research friendly library Kodo provides a number of algorithms and building blocks, with which new and experimental algorithms can be implemented and tested. In this paper we introduce potential users to the goals, the structure, and the use of the library. To demonstrate the use of the library we provide a number of simple programming examples. It is our hope that network coding practitioners will use Kodo as a starting point, and in time contribute by improving and extending the functionality of Kodo.

Keywords: Network Coding, Implementation

1 Introduction

First introduced by Ahlswede et al. in [1], network coding has in recent years received significant attention from a variety of research communities. However despite the large interest most current contributions are largely based on theoretical, analytical, and simulated studies and only few practical implementations have been developed. Of which even fewer has openly shared their implementations. In this paper we introduce an open source network coding library named *Kodo*. It is the hope that Kodo may serve as a practical starting point for researchers and students working in the area of network coding algorithms and implementations. At the time of writing the initial release of Kodo supports basic network coding algorithms and building blocks, however by releasing Kodo as open source it is the hope that contributions may lead to additional algorithms being added to Kodo. Before describing the library design and functionality we will first give an overview of the motivations and goals behind the library.

Flexibility: as the target users of Kodo are researchers working with network coding, one of the key goals behind the design of Kodo has been to create a flexible and extensible Application Programming Interface (API), while providing already functional components for implementers to re-use. To enable

this, the ambition is that Kodo should provide a number of customizable building blocks rather than only a set of complete algorithms. Creating easy to use extension points in the library is vital to provide researchers the opportunity to use/reuse Kodo for their specific use-case or implementation. To achieve this goal Kodo relies on the generic programming paradigms provided by the C++ template system. Through the use of C++ templates implementers may substitute or customize Kodo components without modifying the existing code. This makes experimental configurations easy to create and lowers the entry level for new users.

Ease of use: to the extend possible it is the intent that Kodo should be usable also for researchers with limited programming experience. To achieve this Kodo attempts to provide a simple and clean API hiding the more complicated implementation details. To further support this Kodo will also provide programming language bindings, which will allow users of programming languages other than C++ to use the library functionality. Currently experimental bindings for the Python programming language are provided for a selected part of the API.

Performance: one of the key challenges in the design of Kodo has been to provide the flexibility and ease of use required for research while not significantly sacrificing performance of the library. One of the most active areas for network coding implementations have been investigating and improving the performance of network coding algorithms, see [5],[6],[2]. For this reason Kodo attempts to find a reasonable middle ground between flexibility and performance. In cases where these two requirements does not align, Kodo will typically aim for flexibility and ease of use over performance. The reason for this is to keep the library agile, and that high performance implementations should easily be derived from Kodo source code.

Testing: to avoid and catch as many problems as early as possible, Kodo is tested using a number of unit tests. At the time of writing most components in Kodo is delivered with accompanying test cases, and it is the goal that all Kodo components should have a matching unit test. The testing framework used in Kodo is Qt test library (QtTestLib). It was chosen due to the good cross-platform support and ease of use provided by the Qt framework. It is important to note that the Kodo library itself does not have any dependencies on Qt, and that Kodo therefore may be used on platforms not supported by the Qt framework.

Portability: it is the intention to keep Kodo as portable and self-contained as possible. In cases where platform specific dependencies are required, Kodo should provide interfaces which enable easy adaptation to that specific platform.

Contributions: it is the hope that researchers from the networking coding community will join in the development of Kodo and contribute new functionality. Kodo will be released under a research friendly license which allows everybody to contribute and use the library in their research.

Benchmarking: since Kodo is created to facilitate research of various Network Coding (NC) implementations, enabling measurements and monitor-

ing the performance of the implemented algorithms are a priority. Having good methods for benchmarking allows quick comparisons between performance of existing and future algorithms or optimizations. Using the C++ template system users of Kodo may instrument code to collect useful information about algorithms and data structures used. Examples of this are to monitor the number of finite field operations performed by a specific type of encoder or the number of memory access operations performed during decoding. Additionally it allows the maintainers of Kodo to ensure that no modifications or additions to the library results in unwanted performance regressions.

Following the goals and motivation behind the library the following section will introduce the functionality provided by Kodo.

2 Network Coding Support

Due to its promising potential many algorithms and protocols have been suggested to demonstrate and exploit the benefits of network coding. Use cases range over wired to wireless and from the physical to the transport and application layer. In addition to these widely different areas of application, different variants of network coding also exist. Examples of this are inter- and intra-session network coding, where the former variant operates on data from a uniquely identifiable network flow and the latter allows mixing of data from different network flows. One might also differentiate between deterministic and non-deterministic network coding algorithms. These refer to the way a computer node participating in a network coding system operates on the data traversing it. Typically in deterministic algorithms a node performs a "fixed" number of operations on the incoming data, these predetermined operations may be selected based on different information e.g. the network topology. In contrast non-deterministic network coding or random network coding operates on the data in a random or pseudo-random manner. This has certain desirable advantages in e.g. dynamic wireless networks.

The initial release of Kodo supports only a subset of these use-cases, however it is the goal that future versions will eventually support a wide range of different network coding variants. The initial features implemented have been selected to support the creation of digital Random Linear Network Coding (RLNC) systems. RLNC has in recent years received a large amount of interest for use in dynamic networks e.g. wireless mesh networks [3]. Understanding the functionality of a RLNC system will therefore also be helpful to understand the current structure of the Kodo library. In the following we therefore provide a general overview of the supported RLNC components and show the corresponding C++ code used in Kodo. The C++ programming examples will not be explained in detail here, but are there to demonstrate how a RLNC application would look, using Kodo. For details about usage of the library, the programming API and further programming examples we refer the reader to the Kodo project web page.

2.1 Random Linear Network Coding

In RLNC we operate on either finite or infinite streams of binary data. To make the coding feasible we limit the amount of data considered by segmenting it into manageable sized chunks. Each chunk in turn consists of some number g packets, where each packet has a length of d bytes. In network coding terminology such a chunk of binary data, spanning $g \cdot d$ bytes, is called a *generation*.

One commonly used way for constructing generations from a file is illustrated in Figure 1. In this approach a file is divided into N packets, p_1, p_2, $\cdots$ p_N. Packets are then sequentially divided into $M = \frac{N}{g}$ generations, each containing g packets. In Kodo this algorithm is referred to as the *basic partitioning* algorithm.

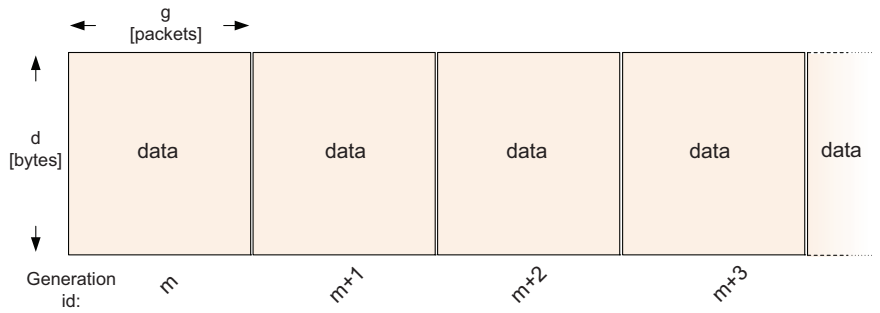


Fig. 1: Basic partitioning algorithm for slicing a file into generations of a certain size.

Besides the basic partitioning algorithm Kodo also supports other partitioning schemes, such as the *random annex code* described in [4]. The following code listing shows how the basic partitioning algorithm is used in Kodo to partition a large buffer.

Listing 1.1: Using the basic partitioning algorithm

```
// Selecting general coding parameters
int generationSize = 16;
int packetSize = 1600;

// Size of the buffer
int bufferPackets = 432141;

// The buffer contaning all the packets of the file
UnalignedBuffer fileBuffer(bufferPackets, packetSize);

// Vector to hold the buffers for each generation
std::vector<ThinBuffer> generationBuffers;
```

```
// Run the basic partitioning, and add the generation
// buffers to the vector
BasicPartitioning basic(generationSize, packetSize);
basic(fileBuffer.dataBuffer(), std::back_inserter(
    generationBuffers));
```

Listing 1.1 Initially we specify the target generation size and the desired packet size for each generation. The *fileBuffer* object represents the file, containing a large amount of packets to be distributed over different generations. This is achieved in the last line by passing file buffer to the basic partitioning algorithm. This creates a number of generation buffers and inserts them in the vector. Following this we may operate on a single generation at a time.

In addition to selecting the packet size and generation size parameters, implementers are faced with an additional choice before the coding can start, namely which *finite field* to use. In NC data operations are performed using finite field arithmetics. Performing arithmetic operations in finite fields are informally very similar to normal arithmetic with integers, the main difference being that only a finite number of elements exist, and that all operations are closed i.e. every operation performed on two field elements will result in an element also in the field. A simple example is the binary field $\mathbb{F}_2$, which consists of two elements namely $\{0, 1\}$. In the binary field addition is implemented using the bit-wise XOR and multiplication is implemented using the bit-wise AND operator. For NC applications the choice of finite field represents a trade-off between computational cost and efficiency of the coding, i.e. how fast we may generate coded packets versus how likely it is to generate linear dependent and therefore useless packets. Kodo provides several choices of finite field implementations, and also allow external implementations to be used.

After creating the buffers containing the generation data and selecting the finite field to use, the encoding and decoding processes may start. Figure 2 presents an overview of the basic operations performed for each generation in a typical RLNC system.

At the top of Figure 2 the encoder uses the g original source packets to create k encoded packets, where k is larger or equal to g. The k encoded packets are then transmitted via an unreliable channel, causing a number of packet erasures. The job at the decoder side is to collect enough encoded packets to be able to reconstruct the original data. In general this is possible as long as the decoder receives g or more encoded packets. Note, that in RLNC the number of encoded packets, k, is not fixed which means that the encoder can continuously create new encoded packets if needed. For this reason this type of coding is often referred to as *rate-less*. Also shown in the figure is one of the most significant differences between NC and traditional schemes, namely the re-coding step shown in the bottom right of Figure 2. In contrast to e.g. traditional error correcting codes, NC based codes are not necessarily end-to-end codes, but allow intermediate nodes between a sender and receiver to re-code and forward data, instead of pure forwarding.

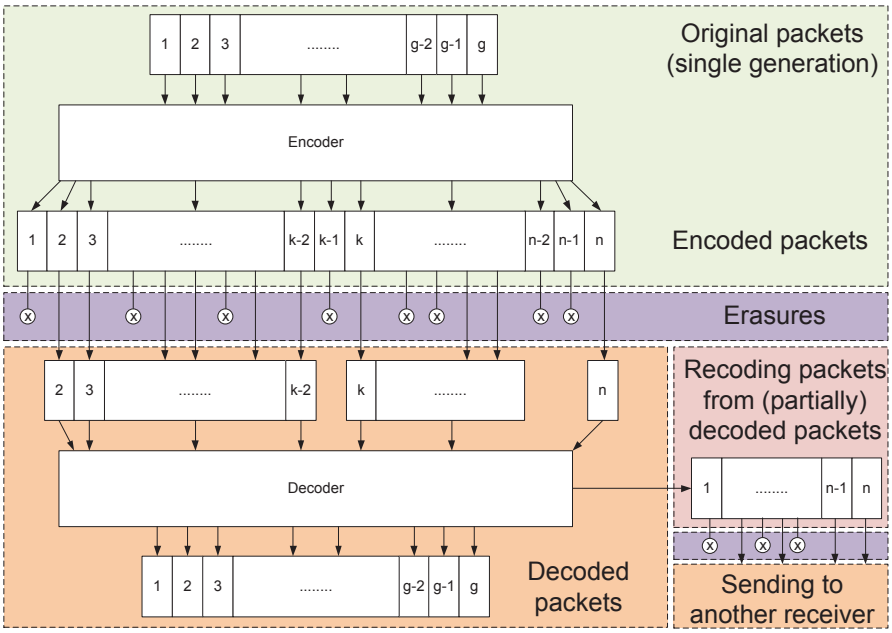


Fig. 2: Network coding system overview.

In Kodo the encoding and decoding step for a single generation may be performed using the following example code:

Listing 1.2: Encoding and decoding data from a single generation

```
int generationSize = 16;
int packetSize = 1600;

// Create an encoder and decoder object
Encoder encoder(generationSize, packetSize);
Decoder decoder(generationSize, packetSize);

// Fill the encoder data buffer with data
fillbuffer(encoder.packetBuffer().buffer(), encoder.
    packetBuffer().size());

// Construct a vector generator
typedef DensityGenerator<Binary, RandMT> DensityGenerator;
DensityGenerator generator(0.5);

int vectorSize = Binary::sizeNeeded(generationSize);

// Make a buffer for one packet and one vector
UnalignedBuffer packet(1, packetSize);
```

```
UnalignedBuffer vector(1, vectorSize);

// Loop as long as we have not decoded
while( !decoder.isComplete() )
{
    memset(packet[0], '\0', packetSize);
    memset(vector[0], '\0', vectorSize);

    // Fill the encoding vector
    generator.fillVector(vector[0], generationSize);

    // Create an encoded packet according to the vector
    encoder.encode(packet[0], vector[0]);

    // For a simulation this would be where the erasures
    // channel could be implemented

    // Pass the encoded packet to the decoder
    decoder.decode(packet[0], vector[0]);
}
```

Listing 1.2 : The source code is a slightly modified excerpt from one of the unit tests, testing the encoder and decoder implementations of Kodo. Initially the coding parameters are specified and some buffers are prepared. We then setup a *vector generator*. The vector generator is responsible for creating the encoding vectors, in this case we are using a *density generator* which creates encoding vectors according to some specified density (i.e. number of non-zero elements in the vector). Kodo contains other types of vector generators e.g. for creating systematic encoding vectors. You may notice the word, *binary*, present in the vector generator, this denotes the finite field we are using. Binary refers to the $\mathbb{F}_2$ binary field. Kodo contains a number of different finite field implementations for different field sizes. Following this the code loops passing encoded packets from the encoder into the decoder until the decoder reports that decoding has completed.

The two source code examples shown demonstrates the code needed to implement an operational RLNC application in less than 60 lines of code. This completes the overall introduction of the Kodo functionality, to get an in-depth view on the possibilities with Kodo visit our project website (which may be found at `www.cone-ftp.dk`.

3 Conclusion

In this paper we have introduced the Kodo C++ network coding library, the goal of this paper is to provided a general overview of the components of the library and introduce its usage. In the initial release of Kodo a number of goals for the

library has been specified, and it is the hope that these goals and introduction of Kodo presented here, may aid researchers in identifying whether Kodo may be useful for them in their network coding research. As shown in this paper Kodo currently support development of basic RLNC schemes and many general NC features are still missing. However, it is the aspiration that by releasing Kodo as open source, researchers will contribute to strengthen the capabilities and usefulness of the project.

4 Acknowledgments

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

References

1. Ahlswede, R., Cai, N., Li, S.Y.R., Yeung, R.W.: Network information flow. IEEE Transactions on Information Theory 46(4), 1204–1216 (July 2000), http://dx.doi.org/10.1109/18.850663
2. Heide, J., Pedersen, M.V., Fitzek, F.H., Larsen, T.: Network coding for mobile devices - systematic binary random rateless codes. In: The IEEE International Conference on Communications (ICC). Dresden, Germany (14-18 June 2009)
3. Ho, T., Medard, M., Koetter, R., Karger, D.R., Effros, M., Shi, J., Leong, B.: A Random Linear Network Coding Approach to Multicast. IEEE Transactions on Information Theory 52(10), 4413–4430 (October 2006), http://dx.doi.org/10.1109/TIT.2006.881746
4. Li, Y., Soljanin, E., Spasojevic, P.: Collecting coded coupons over overlapping generations. CoRR abs/1002.1407 (2010)
5. Shojania, H., Li, B.: Random network coding on the iphone: fact or fiction? In: Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video. pp. 37–42. NOSSDAV '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1542245.1542255
6. Vingelmann, P., Zanaty, P., F.H.P.Fitzek, Charaf, H.: Implementation of random linear network coding on opengl-enabled graphics cards. In: European Wireless 2009. Aalborg, Denmark (may 2009)

Paper 4

Finite Field Arithmetics for Network Coding

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

Book chapter in "Network Coding: A Hands-on Approach" tentative title, to be published Wiley.

Finite Field Arithmetics for Network Coding

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

Abstract

In Network Coding (NC) nodes in a network operate on the data flows traversing them. The arithmetic operations used are defined within a branch of mathematics known as finite fields or Galois fields. In NC finite field arithmetics are used when performing the three core operations namely: encoding, recoding and decoding. An efficient implementation of finite field arithmetics is therefore an important prerequisite for any NC implementation. In this chapter we will introduce the basic finite field theory and describe several ways finite fields may be implemented and used in software.

1 Finite Fields

The purpose of this section is to introduce the reader to the basic theory of finite fields, this is done to assist in the understanding of the software algorithms presented in the following sections. We first give the formal definition of a field along with the axioms governing the operations allowed [2]:

A field is a set $\mathbb{F}$ of at least two elements, where the operations addition (denoted $+$) and multiplication (denoted $\cdot$) are defined, and for which the following axioms are satisfied:

- The set $\mathbb{F}$ forms an abelian group (whose identity is called 0 i.e. $a+0=a$) under the operation $+$. Where the definition of an abelian group states that $a+b=b+a$.
- The set $\mathbb{F}^* = \mathbb{F} - \{0\}$, i.e. the non-zero elements of $\mathbb{F}$ forms an abelian group (whose identity is 1 i.e. $a \cdot 1 = a$) under the operation $\cdot$.
- Distributive law: For all $a, b, c \in \mathbb{F}$ we have, $(a+b) \cdot c = (a \cdot c) + (b \cdot c)$.

By the definition of a field it can be verified that the elements of familiar number systems $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ etc. indeed form fields. If the set $\mathbb{F}$ contains a finite number of elements the field is said to be *finte*. When a field is finite it essentially means that any arithmetic operations performed on two elements in the field will result in an element also in the field.

1.1 Field Operations

Besides the two mentioned field operations (addition and multiplication) we also define the inverse operations namely subtraction and division.

Subtraction of two field elements can be defined in terms of addition, if $a, b \in \mathbb{F}$ then $a - b = a + (-b)$, where $-b$ is the unique field element in $\mathbb{F}$ such that $b + (-b) = 0$ ($-b$ is called the negative of b).

Division can be defined in terms of multiplication: if $a, b \in \mathbb{F}$ then $a/b = a \cdot (b^{-1})$, where b^{-1} is the unique field element in $\mathbb{F}$ such that $b \cdot b^{-1} = 1$ (b^{-1} is called the inverse of b).

As mentioned previously a finite field extends the definition of a field by specifying, that the operations addition and multiplication including their inverse subtraction and division, performed on the field elements must result in an element within the field. For integers this property is achieved using modulo arithmetic. Informally modulo arithmetics can be thought of as normal arithmetics, except that all results x are replaced by $x \mod p$ [3, p. 862-869], where $\mod$ denotes the modulo operation. The result of the modulo operation is the remainder r, which for integers may be calculated as $r = x - pn$, for some quotient n and prime number p. The possible set of mod-p remainders is therefore given as the set $R_p = \{0, 1, \ldots, p-1\}$. In general any integer i can be mapped onto a chosen field as $i \mod p$, when p is a prime number. As shown in Equation (1) choosing $p = 3$ will result in a finite field with three elements $\{0, 1, 2\}$.

$$\begin{aligned} i &\mod p \\ 0 &\mod 3 = 0 \\ 1 &\mod 3 = 1 \\ 2 &\mod 3 = 2 \\ 3 &\mod 3 = 0 \\ 4 &\mod 3 = 1 \\ &\vdots \end{aligned} \tag{1}$$

1.2 Order, Existence and Uniqueness

The number of elements of a field is also called the fields *order*. We may construct a finite field $\mathbb{F}_q$ of order q if and only if q is a prime power, i.e. $q = p^m$, where p is some prime number and m is a positive integer. We call p the *characteristics* of the field. When $m = 1$ we call $\mathbb{F}_p$ a *prime field*. For $m \leq 2$ we call $\mathbb{F}_{p^m}$ an *extension field*. For any field of order q there exists a single finite field. Although the two fields e.g. with parameters $p = 9, m = 1$ and $p = 3, m = 3$ might seem different they contain the same number of elements i.e. have the

same order (namely 9), and thus the only difference between the two fields is the labeling of the field elements. Two finite fields with the same order q are for this reason called *isomorphic*.

1.3 Prime Fields

As shown when p is a prime number the integers modulo p constitutes a prime field with the elements $\{0, 1, \ldots, p-1\}$. The following examples shows the arithmetic operations in the prime field $\mathbb{F}_{11}$:

- Addition: $5 + 9 = 3$, since $14 \mod 11 = 3$.
- Subtraction: $4 - 10 = 5$, since $-6 \mod 11 = 5$
- Multiplication: $6 * 7 = 9$, since $42 \mod 11 = 9$
- Inversion: $4^{-1} = 3$, since $4 * 3 \mod 11 = 1$
- Division: $5/4 = 4$, since $5 * 3 \mod 11 = 4$

The smallest possible finite field $\mathbb{F}_p$ is the prime field with $p = 2$ also called the *binary field*. In this case the order and characteristic of the finite field are equal, however as we mentioned if using extension fields this is not always the case. The binary field consists of two elements $\{0, 1\}$ and is of particular interest since the binary operations are easily implemented and represented in software and hardware. In the case of the binary fields, addition and multiplication are performed mod-2 and therefore satisfies the following addition and multiplication tables:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Table 1: The finite field $\mathbb{F}_2$ consists of elements 0 and 1 which satisfy the addition($+$) and multiplication($\cdot$) tables.

In Table 1 it can be seen that addition can be implemented using the bitwise XOR ($\wedge$) operator and multiplication the bitwise AND (&) operators found in most programming language. If an implementation would only require operations on 1-bit field elements a similar subtraction and division table could be constructed. Where it would quickly be seen that addition and subtraction are identical over $\mathbb{F}_2$. The division table is identical to the multiplication table, except that division by zero would not be allowed. Performing calculations in the 1-bit binary field would however not yield very efficient implementations, as most hardware does not natively support a 1 bit data type i.e. most computer hardware is optimized for operating on 8,16,32 or 64 bit data types. Applications working with $\mathbb{F}_2$ therefore typically perform computations using vectors

of 1 bit field elements to better utilize the underlying hardware. For different reasons depending on the application it may be desirable to increase the field size a common way of achieving is by extending the field.

1.4 Extension Fields

When implementing finite field operations the number of elements in a field can be increased by using polynomials to represent the field elements. A nonzero polynomial $f(x)$ of degree m over a field $\mathbb{F}_{p^m}$ has the form:

$$f(x) = f_m x^m + \ldots + f_2 x^2 + f_1 x + f_0 \tag{2}$$

Where the coefficients $f_i \in \mathbb{F}_p$ for $0 \leq i \leq m$. Polynomials essentially allow the same arithmetic operations as integers, however when polynomials are used the operations are performed mod-$p(x)$, where $p(x)$ is an irreducible polynomial instead of a prime integer. As with a prime number an irreducible polynomial is one which cannot be factored into products of two polynomials.

Ordinary polynomial addition is performed component-wise e.g. for two polynomials with a maximum degree of k:

$$f(x) = h(x) + g(x) \tag{3}$$

$$f(x) = \sum_{i=0}^{k} (h_i + g_i) x^i \tag{4}$$

In $\mathbb{F}_{p^m}$ we calculate $f(x) + g(x)$ as $f(x) + g(x) \mod p(x)$. This uses the usual component-wise addition as given in Equation (4), the only difference is that the coefficient sum is modulo p i.e. $h_i + g_i \mod p$.

For ordinary polynomial multiplication, the coefficients of $f(x) = h(x)g(x)$ are determined by convolution, the resulting polynomial $f(x)$ is of degree = deg(h)+deg(g):

$$f_i = \sum_{j=0}^{i} h_j g_{i-j} \tag{5}$$

The product $h(x)g(x)$ in $\mathbb{F}_{p^m}$ can be found by first multiplying $h(x)$ and $g(x)$ using ordinary polynomial multiplication. Then ensuring that the resulting polynomial $f(x)$ has degree $< m$ by reducing it modulo $p(x)$. The modulo operation can be implemented as polynomial long division and then taking the remainder. As for polynomial addition we must also ensure that all resulting coefficients are elements in $\mathbb{F}_p$ by reducing them modulo p. It can be shown that we may find irreducible polynomials for any p and m [6, p.28].

Of particular interest are binary extension fields i.e. $\mathbb{F}_{2^m}$. These fields are very common as the binary representation allows efficient implementations in either software of hardware.

1.5 Binary Extension Field

In the binary extension field all field elements may be represented as binary polynomials of degree at most $m-1$:

$$\mathbb{F}_{2^m} = \{f_m x^m + \ldots + f_2 x^2 + f_1 x + f_0 : f_i \in \{0,1\}\} \tag{6}$$

As an example consider the field given by $\mathbb{F}_{2^3}$ in Table 2, in this case the field will consist of $2^3 = 8$ polynomial elements of degree $< m$.

$GF2^3$		
Polynomial	Binary	Decimal
0	000	0
1	001	1
x	010	2
$x+1$	011	3
x^2	100	4
x^2+1	101	5
x^2+x	110	6
x^2+x+1	111	7

Table 2: Field elements of the Galois field $GF(2^3)$ in polynomial and binary representation.

In the binary extension field all polynomial elements can be represented as m bit binary numbers. It is important to notice the correspondence between the binary and polynomial representation. The bits from left to right are the coefficients of the powers of x in increasing order. Table 3 shows the direct mapping for a polynomial in $\mathbb{F}_{2^8}$:

1 1 0 1 0 0 1 1
$1 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot 1$
$x^7 + x^6 + x^4 + x + 1$

Table 3: Example mapping between polynomial and binary representation for a polynomial in $\mathbb{F}_{2^8}$.

Understanding the connection between the different representations will be important to understand how finite fields are implemented in software. In general we can construct $\mathbb{F}_2$ extension fields containing any 2^m number of elements where $m \geq 1$. In an implementation the field size can be chosen so that the polynomial representation will fit into an available data type e.g. $\mathbb{F}_{2^8}$ can typically be represented in an unsigned char. For this reason fields of $2^8, 2^{16}, 2^{32}$ elements are common choices.

In the following example it is shown how to perform arithmetic operations on two polynomials from Table 2. The irreducible polynomial $p(x) = x^3 + x + 1$

is used to ensure that all calculations are within $\mathbb{F}_{2^3}$. Note, that coefficient operations are still performed in $\mathbb{F}_2$ we can therefore use the addition table in Table 1 when adding coefficients and subsequently Equation 3.

- Addition: $(x^2+x+1)+(x^2+1) = x$, since $x \mod (x^3+x+1) = x$.
- Subtraction: $(x)-(x^2+x) = x^2$, since $x^2 \mod (x^3+x+1) = x^2$.
- Multiplication: $(x^2+x)\cdot(x+1) = 1$, since $(x^3+x) \mod (x^3+x+1) = 1$.
- Inversion: $x^{-1} = x^2+1$, since $x\cdot(x^2+1) \mod (x^3+x+1) = 1$.
- Division: $x^2/x = x$, since $x^2\cdot(x^2+1) \mod (x^3+x+1) = x$.

As mentioned the modulo operation may be implemented using polynomial long division, taking the remainder. As an example the following computes the polynomial long division of $(x^3+x) \mod (x^3+x+1)$ shown in the multiplication example from above:

$$
\begin{array}{rl|lll}
 & & & & 1 \\
\cline{3-5}
x^3+x+1 & & x^3 & +x & \\
 & & x^3 & +x & +1 \\
\cline{3-5}
 & & & & 1
\end{array}
\begin{array}{l}
\triangleleft \text{ qoutient} \\
 \\
 \\
\triangleleft \text{ remainder}
\end{array}
$$

In addition to the polynomial, binary and decimal representation of the binary field elements, shown in Table 2, we may in certain cases be able to use an alternative representation utilizing two useful properties of binary extension fields, namely primitive polynomials and primitive elements.

1.5.1 Primitive Polynomials

For an irreducible polynomial $p(x)$ defined in $\mathbb{F}_{2^m}$ we investigate the result of $x^j \mod p(x)$, where $j = 0 \rightarrow \infty$. We will find that the modulo operation will only produce a finite number of elements e.g. for $p(x) = x^3+x+1$ we obtain:

$$
\begin{aligned}
x^0 &\mod p(x) = 1 && (7)\\
x^1 &\mod p(x) = x && (8)\\
x^2 &\mod p(x) = x^2 && (9)\\
x^3 &\mod p(x) = x + 1 && (10)\\
x^4 &\mod p(x) = x^2 + x && (11)\\
x^5 &\mod p(x) = x^2 + x + 1 && (12)\\
x^6 &\mod p(x) = x^2 + 1 && (13)\\
x^7 &\mod p(x) = 1 && (14)\\
x^8 &\mod p(x) = x && (15)\\
&\vdots
\end{aligned}
$$

As shown in Equation (7) to (15) we obtain only a limited number remainders from the modulo operation. The number of unique field elements generated by a polynomial is denoted the polynomials *period*. In general we say that the period is the smallest nonzero number P such that:

$$x^P \equiv 1 \mod p(x) \tag{16}$$

Where the maximum period which may be obtained for a polynomial of degree m is given as:

$$P_{max} = 2^m - 1 \tag{17}$$

If a polynomial has a period P_{max} we it denote it as a *primitive polynomial*. Correspondingly any element α for which increasing powers generates all nonzero elements of a given field is called a *primitive element*. Note, that for all primitive polynomials, x is a primitive element. All primitive polynomials are irreducible however not all irreducible polynomials are primitive. This means that choosing an arbitrary irreducible polynomial we may risk that we cannot generate all field elements as the element x is raised to some power j. An example of an irreducible polynomial which is not primitive is $p(x) = x^4 + x^3 + x^2 + x + 1$, where $P_{max} = 2^4 - 1 = 15$:

$$x^0 \mod p(x) = 1 \tag{18}$$
$$x^1 \mod p(x) = x \tag{19}$$
$$x^2 \mod p(x) = x^2 \tag{20}$$
$$x^3 \mod p(x) = x^3 \tag{21}$$
$$x^4 \mod p(x) = x^3 + x^2 + x + 1 \tag{22}$$
$$x^5 \mod p(x) = 1 \tag{23}$$
$$x^6 \mod p(x) = x \tag{24}$$
$$\vdots$$

In this case $p(x)$ does not generate the P_{max} unique elements and is therefore not primitive. For more information on this topic see [11, p.25-54].

We will refer to the x^j as the power representation of the corresponding polynomial element. The advantage of primitive polynomials and elements is that using the power representation, the product or division of two field elements is reduced to summing or subtracting the exponent values modulo $2^m - 1$, where m is the degree of the primitive polynomial. This may in many cases be computed faster than a direct multiplication or division of two binary polynomials. This is shown for multiplication in the following example, using $p(x) = x^3 + x + 1$ from above:

$$a(x) = (x^2 + 1) \cdot (x^2 + x + 1) \mod p(x) \tag{25}$$

Using the mapping from Equation (13) and (12):

$$a(x) = x^6 \cdot x^5 \mod p(x) \tag{26}$$
$$a(x) = x^{6+5 \mod 7} \tag{27}$$
$$a(x) = x^4 \tag{28}$$

Mapping back to polynomial representation using Equation (11):

$$a(x) = x^2 + x \tag{29}$$

Algorithms using finite field arithmetics can in certain cases increase performance by utilizing look-up tables with the mapping between the polynomial representation and the power representation.

2 Finite Fields Implementation

Creating efficient finite fields implementations have been an active research topic for several decades. Many applications in areas such as cryptography, signal processing, erasure coding and now also network coding depend on this research to

deliver satisfactory performance. The efficient implementation of the arithmetic operations in software are complicated for a number of reasons, for example:

- As shown in Section 1 all arithmetic operations must be performed modulo some prime p or irreducible polynomial $p(x)$. For integers and the polynomial coefficients, not in the binary field, the modulo operation is typically performed using an integer division which has a particular high latency on most Central Processing Units (CPUs). Furthermore for polynomials, in all fields, the modulo operation must be performed after multiplication and division (where division is typically implemented using a multiplication with the inverse element) to ensure that the resulting polynomials degree is within the field.
- For prime fields and extension fields where the binary representation of the field elements does not fit the typically available data types e.g. char, short, int, etc. fully utilizing the underlying hardware is typically hard, as most CPUs are designed for $8, 16, 32, 64$ bit data. Therefore in practice operating on data which is not byte-aligned requires bit-masking and bit-shifting making operations slower.

In this section we present a selected number of methods and algorithms which have been proposed for creating efficient finite field implementations. Besides the ones covered here a large amount of additional algorithms exist in current literature. Many of which are optimized for specific use-cases e.g. within the field of cryptography and thus requiring very large field elements (in the order of hundreds of bits per element), and are therefore not directly useful in NC applications, where lower field sizes typically are sufficient [4]. Therefore this section attempts to cover the ones most widely used/suggested for NC applications. Besides the description of the algorithms given here, an open source library containing the corresponding C++ implementation is available at [15].

As we will see finite fields may be implemented in a number of different ways, and in general one can not point out a single superior implementation covering all possible use-cases. Different applications often have different requirements, and for NC some of the factors typically impacting the choice of implementation are:

Field Size: Depending on the network topology we may require a finite field of a certain size, see [4] and [10] for more information.

Memory consumption: On memory constrained devices, algorithms with a limited memory consumption may be a requirement.

Speed: We may wish to optimize the algorithms for speed, in which case we simply want an as fast as possible implementation.

In addition to specific algorithmic optimizations several researchers also deal with platform specific optimizations such as parallelism on multi-core architectures or utilizing the computational capabilities of Graphics Processing

Units (GPUs). An overview of this type of optimizations with focus on NC applications are given in [7].

The algorithms covered here falls in the three categories depicted in Figure 1.

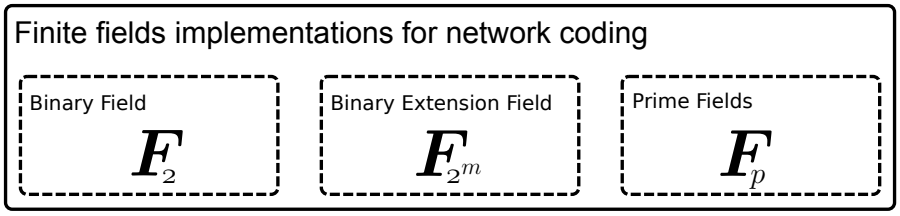
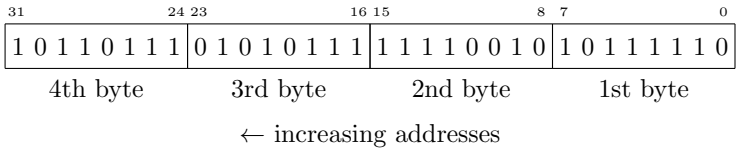


Figure 1: Finite fields commonly used in network coding implementations.

To avoid misunderstandings and make the descriptions more clear we will in the following use terms and definitions:

We assume that an implementation platform has a W-bit architecture, where W is a multiple of 8 (today typically architectures are 8, 16, 32 and 64 bit). The bits of a W-bit word U are numbered from 0 to $W-1$, with the rightmost bit of U denoted bit 0. We will assume Little-endian byte order when addressing multi-byte data, i.e. least significant byte first. As shown in following example for $W = 32$.



The following symbols will be used to denote different operations on words U and V:

$U \wedge V$	bitwise exclusive or (XOR)
$U \& V$	bitwise and (AND)
$U \gg i$	Right shift U by i positions, with the i high-order bits set to 0
$U \ll V$	Left shift U by i positions, with the i low-order bits set to 0
$U \% V$	Compute the modulo between U and V

2.1 Binary Field Implementations

The binary field $\mathbb{F}_2$ only consisting of the two elements $\{0, 1\}$ has been widely used, not only in network coding implementations but also various other fountain code based systems [12, 14]. The use of the binary field is in many cases advantageous since it allows computationally very efficient implementations. When working in the binary field we may utilize the underlying hardware perfectly by operating on vectors of field elements at a time. As an example we

may consider an 8 bit data type (e.g. an unsigned char) as a vector of eight field elements, i.e. with each bit representing an element with value 0 or 1. Pairwise adding two vectors of eight elements may then be implemented using a single XOR operation of two unsigned char variables. Recall from Section 1.3 that all operations in $\mathbb{F}_2$ may be implemented either using the bitwise XOR or AND operators. These operations may typically be performed using a single fast CPU instruction yielding very fast implementations. Network coding algorithms using this field can be found in [9].

Listing 1: Addition and subtraction in $\mathbb{F}_2$

```
def binary_add_subtract(a,b):
    """
    Add or subtracts two vectors of 1 bit field
    elements packed in two W-bit words.

    a, the first field element
    b, the second field element
    """
    return a ^ b
```

Listing 2: Multiplication in $\mathbb{F}_2$

```
def binary_multiply(a,b):
    """
    Multiplies two vectors of 1 bit field
    elements packed in two W-bit words.

    a, the first field element
    b, the second field element
    """
    return a & b
```

Notice, we did not show an algorithm for division here. The reason is that division is only defined for non-zero elements in our field, i.e. no division by zero allowed. Since, we are working in the binary field the only non-zero element is 1, where $1/1 = 1$.

In practice when using the binary field operations are typically performed on vectors of elements. We may readily

2.1.1 Example

As mentioned to better match the supported operations of the underlying hardware when working in $\mathbb{F}_2$ we typically do calculations on vectors of multiple 1 bit elements. As an example if using a $W = 32$ bit architecture using a 32 bit data type would allow adding, subtraction or multiplying up to 32 field elements at a time using a single $\oplus$ or & operation. The following examples show these operations for a number of $W = 8$ bit vectors.

- Addition: $11011001 \oplus 01011011 = 10000010$.

- Subtraction: $01001101 \oplus 00011001 = 01010100$
- Multiplication: $11001011 \& 01001010 = 01001010$

Notice we have omitted division for the reasons explained in Section 2.1.

2.2 Binary Extension Field Implementations

In NC the binary extension field is one of the most commonly used. This is mainly driven by that fact that it enables the use of higher field sizes required by some network topologies, while still allowing reasonably efficient implementations. As we will see in the following choosing an extension field also introduces more complicated algorithms for the multiplication and division operations, when compared to the simple binary field. This added complexity is mainly introduced by the polynomial representation used to create the extension field. To combat this the common strategy is to use look-up tables to fetch precomputed results of the division or multiplication operations. While this technique in most cases can outperform algorithms calculating the results "on the fly", its performance becomes largely platform dependent e.g. on CPU cache sizes, memory prefetching, etc [5]. Furthermore on some resource constrained devices e.g. sensor boards using several kB for a look-up table may be an unacceptable requirement.

In the following we will first provide algorithms for directly computing the result of the different arithmetic operations, we will refer to these as "online" algorithms. Following this we will present a number methods for implementing multiplication and division using precomputed look-up tables, we will refer to these as "offline" algorithms.

2.2.1 Online Addition and Subtraction

As previously shown in Section 1.4 addition and subtraction using the polynomial representation is done element-wise for the coefficients of the polynomials, and since the coefficients in the binary extension field are members of the binary field $\mathbb{F}_2$, this means that addition and subtraction may be implemented using the XOR operation i.e.:

Listing 3: Addition and subtraction in $\mathbb{F}_{2^m}$

```
1 def binary_extension_add_subtract(a, b):
2     """
3     Add or subtracts two W-bit field
4     elements.
5
6     a, the first field element
7     b, the second field element
8     """
9     return a ^ b
```

As the degree of the resulting polynomial $c(x)$ cannot exceed the degree of the chosen prime polynomial, no further computations are needed.

2.2.2 Example

Lets consider a $W = 8$ bit architecture with the two polynomials $a(x) = x^7 + x^6 + x^2$ and $b(x) = x^7 + x^5 + x^3 + x^2$ with the binary representation of 11000100 and 10101100 respectively.

- Addition or subtraction: $11000100 \oplus 10101100 = 01101000$.

The result may be confirmed by adding the two polynomials directly:

$$\begin{aligned} c(x) &= (x^7 + x^6 + x^2) + (x^7 + x^5 + x^3 + x^2) && (30)\\ &= (1 \oplus 1)x^7 + x^6 + x^5 + x^3 + (1 \oplus 1)x^2 && (31)\\ &= x^6 + x^5 + x^3 && (32) \end{aligned}$$

Where $x^6 + x^5 + x^3$ has the expected binary representation 01101000.

2.2.3 Online Multiplication and Division

Compared to the straightforward and simple implementation of addition and subtraction the multiplication and division algorithms require significantly more computations, as shown in the following algorithm. This also illustrates why using precomputed look-up tables often increases performance.

The basic idea behind the following algorithm is to perform multiplication one factor at a time, while ensuring that the resulting polynomial in each step has a degree $< m$. As an example given the two polynomials $a(x) = x^3 + x + 1$, $b(x) = x^2 + 1$ and an irreducible polynomial $p(x) = x^5 + x^2 + 1$, we want to compute the result $c(x) = a(x) \cdot b(x)$:

$$\begin{aligned} c(x) &= (x^3 + x + 1) \cdot (x^2 + 1) && (33)\\ &= x^5 + (1 \oplus 1)x^3 + x^2 + x + 1 && (34)\\ &= x^5 + x^2 + x + 1 && (35) \end{aligned}$$

Since degree of $c(x)$ equal m, we perform the modulo operation using the irreducible polynomial $p(x)$

$$\begin{aligned} c(x) &= x^5 + x^2 + x + 1 \mod p(x) && (36)\\ c(x) &= x && (37) \end{aligned}$$

To implement this in practice we need the following two operations:

Multiplication with x: We perform the multiplication of the two polynomials using three iterations of the algorithm equivalent to the following:

$$c(x) = \underbrace{(x^3 + x + 1) \cdot (x^2 \cdot 1)}_{\text{interation 3}} + \underbrace{(x^3 + x + 1) \cdot (x^1 \cdot 0)}_{\text{iteration 2}} + \underbrace{(x^3 + x + 1) \cdot (x^0 \cdot 1)}_{\text{iteration 1}} \quad (38)$$

A key part to understanding the algorithm is to notice that in each iteration we multiply $a(x)$ with an increasing power of x from $b(x)$. However, as shown in "interation 2" we only use the result if the corresponding coefficient $b(x)$ is non-zero. Using this approach we perform the multiplication by iterating though the binary representation of $b(x)$, while in every step multiplying $a(x)$ with x and if necessary reducing the result modulo $p(x)$. Where the multiplication with x is equivalent to a right bit-shift e.g. taking $a(x)$ from the example we see that:

$$\underbrace{(x^3+x+1)}_{00001011} \rightarrow \underbrace{(x^3+x+1)\cdot x}_{00001011 \ll 1} \rightarrow \underbrace{x^4+x^2+x}_{00010110} \tag{39}$$

Multiplying $a(x)$ with x^2 is equivalent to performing two bit-shifts and so forth.

Modulo reduction with p(x): during each iteration of the algorithm we have to ensure that the degree of the resulting polynomial stays below m. In the cases where when the degree of the result polynomial reaches m we must perform the modulo reduction with the irreducible polynomial $p(x)$. In the example this occurs after "iteration 3". From the example, we have the desired result:

$$c(x) = x^5 + x^2 + x + 1 \mod x^5 + x^2 + 1 \tag{40}$$

$$= x \tag{41}$$

As both polynomials in this case have degree m the modulo operation may be performed using a single subtraction. We know from Section 2.2.1 the subtraction in the binary extension field may be implemented using an XOR between the binary representation of the two polynomials:

$$00100111 \oplus 00100101 = 00000010 \tag{42}$$

The above result may also be verified using polynomial long division, and is equivalent to:

$$\begin{array}{rr|cccc} & & & & & 1 \\ \cline{3-6} x^5+x^2+1 & & x^5 & +x^2 & +x & +1 \\ & & x^5 & +x^2 & & +1 \\ \cline{3-6} & & & & x & \end{array} \quad \begin{array}{l} \triangleleft \text{ qoutient} \\ \\ \\ \triangleleft \text{ remainder} \end{array}$$

Where we can verify that the remainder x indeed has the binary representation 00000010.

The following example[1] shows steps of the algorithm for the input $a = x^2 + x + 1$ and $b = x + 1$:

Listing 4: Online multiplication algorithm for $\mathbb{F}_{2^m}$

```
simple_multiply_print_latex = False

def online_simple_multiply(a, b, p, m):
    """
    Performs the simple online multiply algorithm in the
    2^m extension field

    a, the first polynomial
    b, the second polynomial
    p, the irreducible polynomial
    m, the degree of the irreducible polynomial
    """

    if a == 0 or b == 0:
        return 0

    # Mask to check if the degree is about to reach m
    mask = 1 << (m-1)
    highbit = 0

    # The resulting polynomial
    c = 0

    for i in range(m):

        if b == 0:
            break

        if b & 1:
            c = c ^ a

        highbit = a & mask

        a = a << 1
        b = b >> 1

        if highbit:
            a = a ^ p

    return c
```

[1]Note, that this implementation may not suitable for use in certain cryptographic schemes as we stop the for loop as soon as possible, this may make the code vulnerable for timing attacks.

2.2.4 Example

The following shows the iterations of the algoritm using the example given in Equation 33, where $a(x) = x^3 + x + 1$, $b(x) = x^2 + 1$ and $p(x) = x^5 + x^2 + 1$. The input to the algorithm is the decimal representation of the polynomials as shown in Table 2.

Initialization:

$$a = 11 \qquad b = 5 \qquad c = 0 \qquad highbit = 0 \tag{43}$$
$$p = 37 \qquad m = 5 \qquad mask = 16 \tag{44}$$

Interation 1:

$$a = 22 \qquad b = 2 \qquad c = 11 \qquad highbit = 0 \tag{45}$$
$$p = 37 \qquad m = 5 \qquad mask = 16 \tag{46}$$

Interation 2:

$$a = 9 \qquad b = 1 \qquad c = 11 \qquad highbit = 1 \tag{47}$$
$$p = 37 \qquad m = 5 \qquad mask = 16 \tag{48}$$

Interation 3

$$a = 18 \qquad b = 0 \qquad c = 2 \qquad highbit = 0 \tag{49}$$
$$p = 37 \qquad m = 5 \qquad mask = 16 \tag{50}$$

In "iteration 3" we hit the terminating condition b equal 0. We see that the resulting polynomial is $c = 2$, which corresponds to 00000010 binary or $c(x) = x$ as polynomial and matches our calculations in Equation 37.

2.2.5 Online Division Algorithm

We now define the algorithm used for division, to do this we have to recall the definition presented in Section 1.1. This definition states that division may be implemented in terms of multiplication with the inverse element i.e.

$$c(x) = \frac{a(x)}{b(x)} \tag{51}$$

Will be calculates as:

$$c(x) = a(x) \cdot b(x)^{-1} \tag{52}$$

Since we already have defined an algorithm for multiplication we therefore only need to find an algorithm to determine the inverse of an element in the

binary extension field. A common way to implement this is to use the Extended Euclidean algorithm. The Extended Euclidean algorithm calculates the Greatest Common Divisor (GCD) of two polynomials, but also find two additional polynomials $u(x)$ and $v(x)$ such that:

$$\gcd(a(x), b(x)) = a(x) \cdot u(x) + b(x) \cdot v(x) \tag{53}$$

In the above equation we may find the GCD between two arbitrary polynomials. However, if we choose the fields irreducible polynomial as one of the input polynomials to the algorithm we get the following result:

$$\gcd = a(x) \cdot u(x) + p(x) \cdot v(x) \qquad = 1 \mod p(x) \tag{54}$$

$$\gcd = a(x) \cdot u(x) \qquad = 1 \mod p(x) \tag{55}$$

As $p(x)$ is a irreducible polynomial the GCD will always be 1, furthermore since we are working $\mod p(x)$ any multiple of $p(x)$ will be zero. We therefore see that $u(x)$ will represent the inverse of $a(x)$.

The following algorithm calculates the inverse polynomial by utilizing the fact that the GCD between two polynomials does not change when subtracting one from the other. This algorithm and the Extended Euclidean algorithm from which it is derived is described in more detail in Appendix A.

Listing 5: Online inverse algorithm for $\mathbb{F}_{2^m}$

```
def online_simple_inverse(a, p):
    """
    Finds the inverse of a polynomial a(x) in
    the 2^m extension field

    a, the polynomial whos inverse we want
    p, the irreducible polynomial
    """
    if a == 1:
        return 1

    r_large = p
    r_small = a

    y_large = 0
    y_small = 1

    j = 0
    while(r_large != 1):

        j = find_degree(r_large) - find_degree(r_small)

        if j < 0:
            r_large, r_small = r_small, r_large
            y_large, y_small = y_small, y_large

            j = abs(j)

        r_large = r_large ^ (r_small << j)
        y_large = y_large ^ (y_small << j)

    return y_large
```

Notice, the algorithm requires the degree of the resulting polynomials to be know, a suitable function may be derived by iterating over the binary representation of each polynomials. A code listing for such a function may be found Section A.0.17

The division algorithm may now be implemented in terms of the multiplication and inverse algorithms as shown in the following:

Listing 6: Online divide algorithm for $\mathbb{F}_{2^m}$

```
def online_simple_divide(a, b, p, m):
    """
    Divides the two input polynomials and find the resulting
    polynomial in the 2^m extension field

    a, the numerator polynomial
    b, the denominator polynomial
    p, the prime polynomial
    m, the degree of the prime polynomial
    """
    inverse = online_simple_inverse(b,p)
    return online_simple_multiply(inverse, a, p, m)
```

2.2.6 Example

Since the division algorithm is simply composed using the multiplication and inverse algorithm, we will only show an example of the inverse algorithm here. In the following iterations we find the inverse of $a(x) = x^3 + x + 1$ using the irreducible polynomial $p(x) = x^5 + x^2 + 1$. As with the multiplication the input to the algorithm is specified using the decimal representation of the polynomials:

Initialization:

$$r\_large = 37 \quad y\_large = 0 \quad a = 11 \quad p = 37 \tag{56}$$
$$r\_small = 11 \quad y\_small = 1 \quad p = 37 \tag{57}$$

Iteration 1:

$$r\_large = 9 \quad y\_large = 4 \quad a = 11 \quad p = 37 \tag{58}$$
$$r\_small = 11 \quad y\_small = 1 \quad p = 37 \tag{59}$$

Iteration 2:

$$r\_large = 2 \quad y\_large = 5 \quad a = 11 \quad p = 37 \tag{60}$$
$$r\_small = 11 \quad y\_small = 1 \quad p = 37 \tag{61}$$

Iteration 3:

$$r\_large = 3 \quad y\_large = 21 \quad a = 11 \quad p = 37 \tag{62}$$
$$r\_small = 2 \quad y\_small = 5 \quad p = 37 \tag{63}$$

Iteration 4:

$$r\_large = 1 \quad y\_large = 16 \quad a = 11 \quad p = 37 \tag{64}$$
$$r\_small = 2 \quad y\_small = 5 \quad p = 37 \tag{65}$$

In "iteration 4" we hit the terminating condition $r\_large = 1$, in this case the algorithm will have calculates the inverse polynomial in the variable y_large. Which in this case contains the decimal value 16 which represents the polynomial $u(x) = x^4$. We may now check that this indeed is the inverse of $a(x)$, such that:

$$a(x) \cdot u(x) = 1 \mod p(x) \tag{66}$$

We check by inserting our two polynomials:

$$\tag{67}$$
$$(x^3 + x + 1) \cdot (x^4) = 1 \mod p(x) \tag{68}$$
$$x^7 + x^5 + x^4 = 1 \mod p(x) \tag{69}$$
$$1 = 1 \tag{70}$$

The step given in Equation (69)(70) is calculated by reducing the polynomial $x^7 + x^5 + x^4$ by the irreducible polynomial $p(x) = x^5 + x^2 + 1$. We may verify this using polynomial long division:

$$\begin{array}{rl|lll} & & & x^2 & +1 \quad \triangleleft \text{ qoutient} \\ \cline{3-5} x^5 + x^2 + 1 & x^7 & +x^5 & +x^4 & \\ & x^7 & & +x^4 & +x^2 \\ \cline{2-4} & & x^5 & & +x^2 \\ & & x^5 & & +x^2 \quad +1 \\ \cline{3-5} & & & & 1 \quad \triangleleft \text{ remainder} \end{array}$$

2.2.7 Full Multiplication Table

As seen in the previous section, multiplication of two elements require a significant number of operations. To avoid this we may instead precompute the results and at runtime use a lookup table to find the results. In this way we may reduce the number of operations needed per multiplication at the cost of a high memory consumption. The lookup table may be computed using the iterative multiplication and division algorithms given in the previous section.

Listing 7: Creates a full multiplication and division table for $\mathbb{F}_{2^m}$

```
def create_full_table(p,m):
    """
    Precomputes and creates two lookup table for multiplying
    and dividing elements in a given 2^m binary extension field.

    p, the irreducible polynomial used
    m, the degree of the irreducible polynomial
    """

    # The number of elements in the field
    order = 1 << m

    # Allocate the two tables
    multtable = [0]*order*order
    divitable = [0]*order*order

    for i in range(order):
        offset = i * order

        for j in range(order):
            multtable[offset+j] = online_simple_multiply(i, j, p, m)

            if j == 0: # Cannot divide by zero
                continue

            divitable[offset+j] = online_simple_divide(i, j, p, m)


    return (multtable, divitable)
```

Multiplying or dividing two field elements may now be computed using a look-up into either the multiplication or division table, as shown in Listing 8.

Listing 8: Example look-up function for the full multiplication or division tables created in $\mathbb{F}_{2^m}$

```
def multdiv_full_table(a,b,t,m):
    """
    Multiplies/divides two binary extension field elements using the
    precompute lookup table.
    For each element there are 2^m results, this is used to
    index into the table

    a, the first polynomial
    b, the second polynomial
    t, the precomputed lookup table
    m, the degree of the irreducible polynomial
    """
    return t[(a << m) + b]
```

Memory Consumption: As seen the full multiplication and division tables have a very attractive complexity, requiring only a single bit-shift, addi-

tion and memory lookup to perform either a multiplication or division. However, the low complexity comes with a high memory consumption. Depending on the chosen field $\mathbb{F}_{2^m}$ a table will contain $2^m \cdot 2^m = 2^{2m}$ elements, where each element consumes m bits storage. Computing the total storage for both a multiplication and division table we get:

$$\mathrm{m}_{\mathrm{full}} = 2^{2m} \cdot \frac{m}{8} \cdot 2 \text{ [B]} \tag{71}$$

2.2.8 Log and Anti-Log Table

On way to reduce the memory footprint of the full table based multiplication and division is to construct the lookup table utilizing the power representation of the finite field elements as introduced in Section 1.5.1. This type of implementation is in literature often called the Log/AntiLog method. In the following example we will produce such a lookup table for primitive polynomial $p(x) = x^3 + x + 1$. From Section 1.5.1 we have the following possible representations for the field elements:

Power	Polynomial	Binary	Decimal
0	0	000	0
x^0	1	001	1
x^1	x	010	2
x^2	x^2	100	4
x^3	$x+1$	011	3
x^4	x^2+x	110	6
x^5	x^2+x+1	111	7
x^6	x^2+1	101	5

Table 4: Different representations of the elements of the finite field $\mathbb{F}_{2^3}$.

This basic idea is to create a mapping from the binary/decimal representation to the exponent in the power representation and from the exponent in the power representation back. In our chosen example $\mathbb{F}_{2^3}$ these are shown in Table 5.

index	0	1	2	3	4	5	6	7
Log	-	0	1	3	2	6	4	5
AntiLog	1	2	4	3	6	7	5	-

Table 5: Logarithmic tables for the finite field $\mathbb{F}_{2^3}$.

Using Table 5 we see that the Log table maps between decimal value of an element and its corresponding exponent in the power representation i.e. such that:

$$\text{Log}[5] = 6 \tag{72}$$

Which we can verify using Table 4 which shows that the decimal value 5 has the corresponding power representation x^6. Correspondingly the AntiLog table maps from the exponent value in the power representation to the decimal representation:

$$\text{AntiLog}[6] = 5 \tag{73}$$

This may now be utilized by using the values in Table 5, such that multiplication and division becomes:

$$\begin{aligned} 5 * 3 &= \text{AntiLog}[\text{Log}[5] + \text{Log}[3] \mod 7] && (74)\\ &= \text{AntiLog}[6 + 3 \mod 7] && (75)\\ &= \text{AntiLog}[2] = 4 && (76) \end{aligned}$$

$$\begin{aligned} 2 * 4 &= \text{AntiLog}[\text{Log}[2] + \text{Log}[4] \mod 7] && (77)\\ &= \text{AntiLog}[1 + 2 \mod 7] && (78)\\ &= \text{AntiLog}[3] = 3 && (79) \end{aligned}$$

$$\begin{aligned} 6/3 &= \text{AntiLog}[\text{Log}[6] - \text{Log}[3] \mod 7] && (80)\\ &= \text{AntiLog}[4 - 3 \mod 7] && (81)\\ &= \text{AntiLog}[1] = 2 && (82) \end{aligned}$$

$$\begin{aligned} 2/7 &= \text{AntiLog}[\text{Log}[2] - \text{Log}[7] \mod 7] && (83)\\ &= \text{AntiLog}[1 - 5 \mod 7] && (84)\\ &= \text{AntiLog}[3] = 3 && (85) \end{aligned}$$

As shown we may find the power representation for the field elements by calculating the $j'th$ power of the primitive element x. In the following code listing we construct the Log and AntiLog tables using the online multiplication algorithm to repeatedly multiply x with itself and finding the corresponding field elements.

Listing 9: Creates the Log and AntiLog tables for $\mathbb{F}_{2^m}$

```
def create_log_table(p,m):
    """
    Creates the Log and AntiLog tables mapping between power
    representation and decimal/binary representation in a
    given 2^m binary extension field.

    p, the primitive polynomial used
    m, the degree of the primitive polynomial
    """

    # The number of elements in the field
    order = 1 << m

    # Allocate the two tables
    log     = [None]*order
    antilog = [None]*order

    # Initial value corresponds to x^0
    power = 1

    for i in range(order-1):

        log[power] = i
        antilog[i] = power

        # 2 is the decimal representation of the polynomial
        # element 'x'
        power = online_simple_multiply(2, power, p, m)

    return (log, antilog)
```

For multiplication we may use the Log and AntiLog tables as follows:

Listing 10: Multiplication using the Log and AntiLog tables for $\mathbb{F}_{2^m}$

```
def multiply_log_table(a,b,log,antilog,m):
    """
    Multiplies two field elements using the
    precompute lookup table.

    a, the first field element
    b, the second field element
    log, the precomputed Log lookup table
    antilog, the precomputed AntiLog lookup table
    m, the degree of the irreducible polynomial
    """
    if a == 0 or b == 0:
        return 0

    # Calculate the exponent sum modulo the field
    # order - 1
    exp_sum = (log[a] + log[b]) % (2^m - 1)

    return antilog[exp_sum]
```

Division using the Log and AntiLog tables are similar to multiplication, but uses subtraction of exponent values instead of addition:

Listing 11: Division using the Log and AntiLog tables for $\mathbb{F}_{2^m}$

```
def divide_log_table(a,b,log,antilog,m):
    """
    Divides two field elements using the
    precompute lookup table.

    a, the numerator field element
    b, the denomiator field element
    log, the precomputed Log lookup table
    antilog, the precomputed AntiLog lookup table
    m, the degree of the irreducible polynomial
    """
    # No division by zero
    assert(b > 0)

    if a == 0:
        return 0

    # Calculate the exponent sum modulo the field
    # order - 1
    exp_sum = (log[a] - log[b]) % (2^m - 1)

    return antilog[exp_sum]
```

Removing modulo operator: A performance bottleneck in the presented algorithm is the use of the modulo operator to reduce the exponent sum. Since the modulo operator typically requires significantly more CPU cycles to perform that e.g. addition and subtraction we may improve perfor-

mance by avoiding its use. Here we assume that the implementation uses a finite precision type system with C/C++ compatible unsigned integer overflow rules. We first notice that if the exponent sum overflows we may detect it in the following way: The exponent sum of two W bit unsigned integers is bound by $s = a + b < 2^{W+1}$, in case of an overflow we have $2^W \leq a + b < 2^{W+1}$ in which case the leading bit in the $W + 1$ bit representation of the sum equals 1, since this high bit cannot be represented in the W bit representation it will be discarded. Discarding the high means that $s < a$ and $s < b$, this can therefore easily be checked.

After detecting the overflow we have to ensure that the modulo reduction is performed correctly. We do this by noticing that if an overflow occurs this is equivalent to subtracting 2^W from the sum. However, since we work $\mod (2^W - 1)$ we have to compensate by adding 1 to the resulting sum. If, no overflow occurs we do not have to perform any operations.

Memory Consumption: As seen both the Log and AntiLog tables will contain a single entry for each element in the used field. In general a $\mathbb{F}_{2^m}$ field has 2^m elements of m bits. Thus the total memory consumption for both the Log and AnitLog tables become:

$$\mathrm{m}_{\log} = 2^m \cdot \frac{m}{8} \cdot 2 \text{ [B]} \tag{86}$$

2.2.9 Extended Log and Anti-Log Table

In the previous section we showed how to construct the Log and AntiLog tables, we also discussed how the modulo operator could be avoid to increase performance. A different approach to handle overflows in the exponent sum is to extend the tables to handle this situation. The following optimization is based on the fact that the maximum exponent is given as:

$$e_{max} = 2^m - 2 \tag{87}$$

Furthermore the minimum exponent is given as

$$e_{min} = 0 \tag{88}$$

Thus, we have the maximum exponent sum as:

$$sum_{max} = (2^m - 2) + (2^m - 2) \tag{89}$$
$$= 2^{m+1} - 4 \tag{90}$$

Likewise the minimum exponent sum can be calculated as:

$$sum_{min} = 0 - (2^m - 2) \tag{91}$$
$$= 2 - 2^m \tag{92}$$

Accepting an increased memory consumption we may extend the AntiLog table to handle all possible sums. The following AntiLog table extends the example given in Table 5 for $p(x) = x^3 + x + 1$:

index	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12
AntiLog	2	4	3	6	7	5	1	2	4	3	6	7	5	1	2	4	3	6	7

Table 6: Extended AntiLog tables for the finite field $\mathbb{F}_{2^3}$.

To understand the mapping recall that we are computing the sum modulo $2^m - 1$. In the example this yields:

$$-6 \mod 7 = 1 \tag{93}$$
$$-5 \mod 7 = 2 \tag{94}$$
$$\vdots \tag{95}$$
$$11 \mod 7 = 4 \tag{96}$$
$$12 \mod 7 = 5 \tag{97}$$

From Equation (93) we see that index -6 should map to same element as index 1, Equation (94) shows index -5 maps to the same element as index 2 and so forth. We can verify these results in Table 6 by checking that e.g. index -6 and 1 both map to the same element. We may now rewrite the examples from Section 2.2.8 without the mod operator:

$$5 * 3 = \text{AntiLog}[\text{Log}[5] + \text{Log}[3]] \tag{98}$$
$$= \text{AntiLog}[6 + 3] \tag{99}$$
$$= \text{AntiLog}[9] = 4 \tag{100}$$

$$2 * 4 = \text{AntiLog}[\text{Log}[2] + \text{Log}[4]] \tag{101}$$
$$= \text{AntiLog}[1 + 2] \tag{102}$$
$$= \text{AntiLog}[3] = 3 \tag{103}$$

$$6/3 = \text{AntiLog}[\text{Log}[6] - \text{Log}[3]] \tag{104}$$
$$= \text{AntiLog}[4 - 3] \tag{105}$$
$$= \text{AntiLog}[1] = 2 \tag{106}$$

$$2/7 = \text{AntiLog}[\text{Log}[2] - \text{Log}[7]] \tag{107}$$
$$= \text{AntiLog}[1 - 5] \tag{108}$$
$$= \text{AntiLog}[-4] = 3 \tag{109}$$

The following function shows how the extended Log and AntiLog tables may be created:

Listing 12: Creates the extended Log and AntiLog tables for $\mathbb{F}_{2^m}$

```
def create_extended_log_table(p,m):
    """
    Creates the extended Log and AntiLog tables mapping between
        power
    representation and decimal/binary representation in a
    given 2^m binary extension field.

    p, the primitive polynomial used
    m, the degree of the primitive polynomial
    """

    # The number of elements in the field
    order = 1 << m

    # Allocate the two tables
    log     = [None] * order
    antilog = [None] * (3 * order - 5)

    # Initial value corresponds to x^0
    power = 1

    # Array offset
    low_offset  = 0
    mid_offset  = low_offset + order - 2
    high_offset = mid_offset + order - 1

    for i in range(order-1):

        log[power] = i
        antilog[mid_offset + i] = power

        # 2 is the decimal representation of the polynomial
        # element 'x'
        power = online_simple_multiply(2, power, p, m)

    # Fill the extended table based on the previously computed
    # results
    for i in range(order-2):
        antilog[low_offset + i]  = antilog[mid_offset + i + 1]
        antilog[high_offset + i] = antilog[mid_offset + i]

    return (log, antilog)
```

Memory Consumption: The Log table size has not changed i.e. it contains 2^m elements of m bits. However, extending the AntiLog table to avoid the modulo operation has increased the number of elements to:

$$\text{AntiLog}_{\text{elements}} = \underbrace{(2^m - 2)}_{negativesums} + \underbrace{(2^{m+1} - 4)}_{positivesums} + \underbrace{(1)}_{zerosum} = (3 \cdot 2^m) - 5 \tag{110}$$

Thus the total memory consumption for both the extended Log and AntiLog tables become:

$$\text{m}_{\text{extlog}} = (2^m + 3 \cdot 2^m - 5) \cdot \frac{m}{8} \text{ [B]} \tag{111}$$

$$= (2^{m+2} - 5) \cdot \frac{m}{8} \text{ [B]} \tag{112}$$

Which is still a significant reduction in memory consumption when compared to the full lookup tables presented in Section 2.2.7.

3 Summary

In the previous sections we have presented a number of different finite field implementations. As shown in [5, 8] which implementation has the best performance for a specific application can vary greatly. Using the algorithms presented here the implementer has a wide variety of choices covering different field sizes, memory consumption and computational complexity. The code listings presented in this chapter demonstrates the different algorithms implemented in the popular Python language [1]. These should form a solid starting point for further experimentation with finite fields. However, if wanted the algorithms presented here are also available as C++ source code at [15]. The source code is freely available for any educational and research related purposes.

References

[1] The python programming language. http://python.org.

[2] *Introduction to Finite Fields*, Lecture Notes in Computer Science. MIT OpenCourseWare, 2005.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[4] O. Geil, R. Matsumoto, and C. Thomsen. On field size and success probability in network coding. In *Arithmetic of Finite Fields*, volume 5130 of *Lecture Notes in Computer Science*, pages 157–173. Springer Berlin, Heidelberg, 2008.

[5] K. M. Greenan, E. L. Miller, and T. J. E. Schwarz. Optimizing galois field arithmetic for diverse processor architectures and applications. In *MASCOTS*, pages 257–266, 2008.

[6] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[7] J. Heide, M. Pedersen, F. Fitzek, and T. Larsen. *Network Coding in the Real World*, pages 87–114. Academic Press, Incorporated, 1 edition, 2011. 2011; 4.

[8] J. Heide, M. Pedersen, F. Fitzek, and M. Medard. On code parameters and coding vector representation for practical rlnc. *I E E E International Conference on Communications*, pages 1 – 5, 2011.

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

[10] 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*, 52(10):4413 –4430, oct. 2006.

[11] S. Lin and D. J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[12] M. Luby. Lt codes. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 271–, Washington, DC, USA, 2002. IEEE Computer Society.

[13] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[14] A. Shokrollahi. Raptor codes. In *IEEE Transactions on Information Theory*, pages 2551–2567, 2006.

[15] Steinwurf ApS. Fifi git repository on github, 2012. http://github.com/steinwurf/fifi.

Appendix

A Extended Euclidean Algorithm

In this section we describe the extended Euclidean algorithm with can be used to find the inverse of integers and polynomials defined over some field $\mathbb{F}_{\mathrm{p}^{\mathrm{m}}}$. The extended Euclidean algorithm is based on the classical Euclidean algorithm for finding the GCD of two numbers. The classical Euclidean algorithm works by using the principle that given two numbers a and b where $a \leq b$ subtracting a from b does not change the GCD. This yields the following identity:

$$gcd(a, b) = gcd(b - qa, a) \tag{113}$$

Equivalent to:

$$gcd(a, b) = gcd(b \bmod a, a) \tag{114}$$

This leads to the following interative algoritm where we find the quotient q for positive integers a and b where $a \leq b$. Here b is divided by a to obtain a quotient q and a remainder r satisfying $r = b - qa$ and $0 \leq r \leq a$. From Equation 113 we know that $gcd(a, b) = gcd(r, a)$, the problem of finding the $gcd(a, b)$ has now been reduced to computing $gcd(r, a)$ where $(r, a) < (a, b)$ this process now continues until the remainder is 0, which immediately yields the result $gcd(0, d)$, where d is the GCD. The following example shows the process for $a = 4, b = 27$.

$$\begin{aligned} gcd(4, 27) &= gcd(27 - 6 \cdot 4, 4) = gcd(3, 4) \\ &= gcd(4 - 1 \cdot 3, 3) = gcd(1, 3) \\ &= gcd(3 - 3 \cdot 1, 1) = gcd(0, 1) \\ &= 1 \end{aligned} \tag{115}$$

The final non-zero remainder is the gcd in this case 1.

The Euclidean algorithm outlined may now be extended to find the integers x, y such that $ax + by = d$, where $d = gcd(a, b)$. The extended Euclidean algorithm can be implemented in a number of different ways. In the following we will outline an algorithm based on the Table Based Method (also know a "magic box"). The algorithm is based on the observation that the remainders computed in the standard GCD algorithm can be expressed as linear combinations of the original integers a and b:

$$r_i = ax_i + by_i \tag{116}$$

So in general we may say:

$$r_{i-1} = ax_{i-1} + by_{i-1} \tag{117}$$
$$r_{i-2} = ax_{i-2} + by_{i-2} \tag{118}$$
$$\vdots \tag{119}$$

From Equation 114 we see that the remainder r_i can be calculated as:

$$r_i = r_{i-2} \mod r_{i-1} \tag{120}$$
$$= r_{i-2} - \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor \cdot r_{i-1} \tag{121}$$

Substituting Equation (117) and (118) into Equation (121) yields:

$$r_i = ax_{i-2} + by_{i-2} - \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor \cdot (ax_{i-1} + by_{i-1}) \tag{122}$$

Which may be rewritten as:

$$r_i = ax_{i-2} + by_{i-2} - ax_{i-1} \cdot \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor - by_{i-1} \cdot \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor \tag{123}$$
$$= a\left(x_{i-2} - x_{i-1} \cdot \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor\right) + b\left(y_{i-2} - y_{i-1} \cdot \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor\right) \tag{124}$$

Comparing Equation (124) to Equation (116) we see that:

$$x_i = x_{i-2} - x_{i-1} \cdot \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor \tag{125}$$
$$y_i = y_{i-2} - y_{i-1} \cdot \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor \tag{126}$$

After finding these identities we can determine the coefficients x_i and y_i for any given remainder r_i. We may also note that for every remainder we must know the results of the previous two steps i.e. to compute r_i for a given input a and b we need to first determine $\{x_{i-2}, x_{i-1}, y_{i-1}, y_{i-2}, r_{i-1}, r_{i-2}\}$.

We first initialize the algorithm by calculating r_0 and r_1 by first setting $x_0 = 0, x_1 = 1, y_0 = 1, y_1 = 0$ for these values solve for r_i using Equation (116).

$$r_0 = ax_0 + by_0 = a \cdot 0 + b \cdot 1 = b \tag{127}$$
$$r_1 = ax_1 + by_1 = a \cdot 1 + b \cdot 0 = a \tag{128}$$

We continue calculating $r_2 = ax_2 + by_2$ using the formulas given in Equation (125) and (126). This process continues until the remainder $r_i = 0$ in which case we have found:

$$r_{i-1} = ax_{i-1} + by_{i-1} = gcd(a, b) \tag{129}$$

Algorithm 1 outlines the pseudo code for the extended Euclidean algorithm for integers. This algorithm uses the extended Euclidean procedure which we have outlined in the previous section starting with Equation (116).

Algorithm 1 Extended Euclidean Algorithm for integers in $\mathbb{Z}$, where $a \leq b$

procedure EXTENDED EUCLIDEAN(a,b)
    $x_0 = 0$, $x_1 = 1$
    $y_0 = 1$, $y_1 = 0$
    $r_0 = b$, $r_1 = a$
    while $r_1 \neq 0$ **do**
        $q = \left\lfloor \frac{r_0}{r_1} \right\rfloor$
        $r = r_0 - r_1 \cdot q$
        $x = x_0 - x_1 \cdot q$
        $y = y_0 - y_1 \cdot q$
        $x_1, x_0 \Leftrightarrow x, x_1$
        $y_1, y_0 \Leftrightarrow y, y_1$
        $r_1, r_0 \Leftrightarrow r, r_1$
    end while

Return r_0, x_0 and y_0
end procedure

The following iterations shows the algorithm for the integers $a = 3$ and $b = 11$.

A.0.10 Initialization

$$a = 3 \qquad b = 11 \qquad x = 0 \qquad y = 0 \qquad q = 0 \tag{130}$$
$$r = 0 \qquad x_0 = 1 \qquad x_1 = 0 \qquad y_0 = 0 \qquad y_1 = 1 \tag{131}$$

A.0.11 Iteration 1

$$a = 2 \qquad b = 3 \qquad x = -3 \qquad y = 1 \qquad q = 3 \tag{132}$$
$$r = 2 \qquad x_0 = -3 \qquad x_1 = 1 \qquad y_0 = 1 \qquad y_1 = 0 \tag{133}$$

A.0.12 Iteration 2

$$a = 1 \qquad b = 2 \qquad x = 4 \qquad y = -1 \qquad q = 1 \tag{134}$$
$$r = 1 \qquad x_0 = 4 \qquad x_1 = -3 \qquad y_0 = -1 \qquad y_1 = 1 \tag{135}$$

A.0.13 Iteration 3

$$a = 0 \qquad b = 1 \qquad x = -11 \qquad y = 3 \qquad q = 2 \tag{136}$$

$$r = 0 \qquad x_0 = -11 \qquad x_1 = 4 \qquad y_0 = 3 \qquad y_1 = -1 \tag{137}$$

After terminating the algorithm we may verify the result i.e. $ax + by = d = gcd(a, b)$, where we have $a = 3$, $b = 11$, $x = 4$, $y = -1$ and $gcd(a, b) = d = 1$:

$$3 \cdot 4 + 11 \cdot -1 = 1 \tag{138}$$

In the following we will use this result to find the inverse of a number in a prime field $\mathbb{F}_p$. To see how this may be achieved we use the following observations. First note that if p is prime then $gcd(a, p) = 1$. This mean that for the extended Euclidean algorithm we have $ax + py = 1$. However, as we are in a finite field we should reduce the coefficients modulo p. This yields $ax + yp = 1 \mod p$. Which is $ax = 1 \mod p$ since the modulo of $y \cdot p \mod p$ always will be zero. From this we see that x must be the inverse of a, as the definition of inverse states that $a \cdot a^{-1} = 1$. Based on these results we see that the following optimizations may be made to Algorithm 1, so that we may use it to find the inverse of a.

- We only need to keep track of the x coefficients. As the equation we solving for is essentially $ax = 1$.
- We can terminate the loop when the remainder equals 1. We may use this as terminating condition as we know $gcd(a, p)$ will always equal 1.

Implementing these optimizations we obtain the following algorithm for finding the inverse of an integer in $\mathbb{F}_{\mathbb{p}}$.

Algorithm 2 Finding inverse integer in $\mathbb{F}_p$

```
1:  procedure INVERSE(a)
2:      x_0 = 0, x_1 = 1
3:      r_0 = p, r_1 = a
4:      while r_0 ≠ 1 do
5:          q = ⌊r_0 / r_1⌋
6:          r = r_0 − r_1 · q
7:          x = x_0 − x_1 · q
8:          x_1, x_0 ⇔ x, x_1
9:          r_1, r_0 ⇔ r, r_1
10:     end while
11: Return  x_0 mod p
12: end procedure
```

The following iterations shows the algorithm for the integers $a = 4$ and $p = 7$.

A.0.14 Initialization

$$a = 4 \qquad b = 7 \qquad x = 0 \qquad q = 0 \tag{139}$$
$$r = 0 \qquad x_0 = 1 \qquad x_1 = 0 \tag{140}$$

A.0.15 Iteration 1

$$a = 3 \qquad b = 4 \qquad x = -1 \qquad q = 1 \tag{141}$$
$$r = 3 \qquad x_0 = -1 \qquad x_1 = 1 \tag{142}$$

A.0.16 Iteration 2

$$a = 1 \qquad b = 3 \qquad x = 2 \qquad q = 1 \tag{143}$$
$$r = 1 \qquad x_0 = 2 \qquad x_1 = -1 \tag{144}$$

After reaching the terminating condition i.e. $a = 1$ we may check that the x found is indeed the inverse of a:

$$a \cdot x \mod p = 1 \tag{145}$$
$$4 \cdot 2 \mod 7 = 1 \tag{146}$$

The final part of this section describes how the above results may be used in an extension field i.e. a finite field $\mathbb{F}_{p^m}$ using polynomial representation for the field elements. As it turns out the exactly same formulas apply for polynomials as for integers. Using polynomial long division to find the quotient and remainder in Algorithm 1 and otherwise performing the same routine would yield the correct answer.

That is for two binary polynomials $a(x)$ and $b(x)$ we may find a GCD for which the following holds:

$$gcd(a(x), b(x)) = gcd(b(x) - c(x) \cdot a(x), a(x)) \tag{147}$$

This holds for any binary polynomial $c(x)$. As with the integers Equation (147) states that subtracting a multiple of $a(x)$ from $b(x)$ does not change *gcd*. As for the integers we may extend this result so that we may using the extended Euclidean algorithm may find an $z(x)$ and $w(x)$ such that:

$$a(x) \cdot z(x) + b(x) \cdot w(x) = d(x) = gcd(a(x), b(x)) \tag{148}$$

Algorithms for the both the Euclidean and extended Euclidean algorithms for polynomials are given in [13, p. 82-83]. However, in these algorithms we must perform an polynomial long division to obtain the needed qoutient and

remainder (denoted q and r in the extenede eucliden algorithm for integers) which in practice is not easily implemented in software. Instead we may use an alternative extended Euclidean algorithm presented here [6, p. 57-58]. The algorithnm utilizes the fact that the division may be replace with a number of subtractions. Recall that the $gcd(a, b) = gcd(b, b - ca)$. In this algorithm only one step of each polynomial long division is performed for each interation of the algorihtm. Essentially this means that in each iteration of the algorithm we perform the long division using a single multiplication followed by a subtraction cancel the highest degree polynomial term. As an example we may consider the two polynomials $g(x) = x^4 + x^2 + 1$ and $f(x) = x^3 + 1$. To cancel the term x^4 we first multiply $f(x)$ with x and subtract from $g(x)$ i.e.:

$$\begin{aligned} r(x) &= g(x) + f(x) \cdot x && (149)\\ &= (x^4 + x^2 + 1) + (x^3 + 1) \cdot x && (150)\\ &= (1 + 1) \cdot x^4 + x^2 + x + 1 && (151)\\ &= x^2 + x + 1 && (152) \end{aligned}$$

Notice, that in the binary field addition and subtraction are identical, thus the use of $+$. Furthermore remember that in the binary field $1 + 1 = 0$ which cancles x^4 in (151). We may now continue gcd algorithm by reducing $f(x)$ by $r(x)$.

$$\begin{aligned} h(x) &= f(x) + r(x) \cdot x && (153)\\ &= (x^3 + 1) + (x^2 + x + 1) \cdot x && (154)\\ &= (1 + 1) \cdot x^3 + x^2 + x + 1 && (155)\\ &= x^2 + x + 1 && (156) \end{aligned}$$

$$\begin{aligned} k(x) &= r(x) + h(x) && (157)\\ &= (x^2 + x + 1) + (x^2 + x + 1) && (158)\\ &= (1 + 1) \cdot x^2 + (1 + 1) \cdot x + (1 + 1) && (159)\\ &= 0 && (160) \end{aligned}$$

Since the last remainder is zero, the algorithm ends with $h(x) = x^2 + x + 1$ as the greates common divisor of $g(x)$ and $f(x)$. We may backtrack the steps in this algorithm to find the coefficients of the Extended Euclidean Algorithm (starting from $h(x)$ which is our gcd).

$$h(x) = f(x) + r(x) \cdot x \tag{161}$$
$$= f(x) + (g(x) + f(x) \cdot x) \cdot x \tag{162}$$
$$= f(x) + g(x) \cdot x + f(x) \cdot x^2 \tag{163}$$
$$= (1 + x^2) \cdot f(x) + x \cdot g(x) \tag{164}$$

Using the outlined *gcd* approach for polynomials we may used the following algorithm:

Algorithm 3 Extended Euclidean Algorithm for polynomials in $\mathbb{F}_{2^p}$, where $deg(a) \leq deg(b)$

```
1: procedure EXTENDED EUCLIDEAN(a,b)
2:     g_0 = 1, g_1 = 0
3:     h_0 = 0, h_1 = 1
4:     u = a, v = b
5:     while u ≠ 0 do
6:         j = deg(u) − deg(v)
7:         if j < 0 then
8:             u ↔ v
9:             g_0 ↔ g_1
10:            h_0 ↔ h_1
11:            j ← −j
12:        end if
13:        u = u + v · x^j
14:        g_0 = g_0 + g_1 · x^j
15:        h_0 = h_0 + h_1 · x^j
16:    end while
17:
18: Return  v, g_1 and h_1
19: end procedure
```

As for the integers we may modify the Extended Euclidean Algorithm to find the inverse of any given polynomial mod $p(x)$ as shown in Algorithm 4.

A.0.17 Find Degree Function

Certain algorithms require that the degree of the polynomials are calculated. The following code listing gives an example implementation of such a function.

Algorithm 4 Inverse Algorithm for polynomials in $\mathbb{F}_{2^p}$

procedure INVERSE(a)
  $g_0 = 1$, $g_1 = 0$
  $u = a$, $v = p$
  while u $\neq 1$ **do**
    $j = deg(u) - deg(v)$
    if $j < 0$ **then**
      $a \leftrightarrow b$
      $g_0 \leftrightarrow g_1$
      $j \leftarrow -j$
    end if
    $u = u + v \cdot x^j$
    $g_0 = g_0 + g_1 \cdot x^j$
  end while

Return g_0
end procedure

Listing 13: Function to determine the degree of arbitrary polynomial in $\mathbb{F}_{2^m}$

```
def find_degree(a):
    """
    Finds returns the degree of the polynomial
    i.e. the number 5 = 101 = X^2 + 1 has degree 2

    a, the polynomial whos degree we wish to find
    """
    degree = 0

    a = a >> 1

    while(a > 0):
        degree = degree + 1
        a = a >> 1

    return degree
```

Paper 5

Network Coding Over The $2^{32} - 5$ Prime Field

Morten Videbæk Pedersen, Janus Heide, Péter Vingelmann
and Frank H. P. Fitzek

IEEE Transactions on Mobile Computing (*in preparation for submission*)

Network Coding Over The $2^{32} - 5$ Prime Field

Morten Videbæk Pedersen[†], Janus Heide[†], Péter Vingelmann[*†], and Frank H. P. Fitzek[†]
[†]Department of Electronic Systems, Faculty of Engineering and Science, Aalborg University, Denmark
[*]Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary

***Abstract*—Creating efficient finite fields implementations has been an active research topic for several decades. Many applications in areas such as cryptography, signal processing, erasure coding and now also network coding depend on this research to deliver satisfactory performance. This paper introduces the initial investigation of utilizing prime fields for network coding applications. First we introduce the algorithms needed to apply prime field arithmetics to arbitrary binary data. After this we present the initial throughput measurements from a benchmark application written in C++. These results are finally compared to different binary and binary extension field implementations.**

I. Introduction

With its introduction by Ahlswede in 2000, Network Coding (NC) has undergone a tremendous evolution, from simple XOR type coding schemes to newer developments such as Random Linear Network Coding (RLNC). NC has shown its potential in a variety of application fields covering sensor networks, satellite networks and peer-to-peer (P2P) networks as a short list of examples. The core idea behind NC schemes is to change the way packets are processed in the network. Without NC, the packets are typically only stored and forwarded at intermediate nodes in the network. With NC, packets are potentially recoded before being forwarded. As shown with the famous butterfly example [1], this approach leads to significant potential gains.

The arithmetic operations performed by the NC nodes in a network are defined within a branch of mathematics known as finite fields or Galois fields. Finite fields define all the commonly used arithmetic operations i.e. addition, subtraction, multiplication and division. These arithmetics operations are used in NC when performing the three core operations namely: encoding, recoding and decoding. Efficient implementations of finite field arithmetics are therefore an important prerequisite for any efficient NC implementation. See [2] for a thorough introduction to the theory of finite fields.

In both software and hardware, finite fields may be implemented in a number of different ways, and in general one cannot point out a single superior implementation covering all possible use-cases [3]. Different applications will often have different requirements. The finite field implementation presented in this paper addresses two requirements commonly seen in NC applications, namely high field sizes and low algorithmic memory consumption.

A. Field Size

Depending on the network topology, a large field size may be required to efficiently realize the communication. As stated by the main theorem of NC given in [4].

Theorem 1 (Main Theorem in Network Coding): Consider a directed acyclic graph $G = (V, E)$ with unit capacity edges, h unit rate sources located on the same vertex of the graph and N receivers. Assume that the value of the min-cut to each receiver is h. Then there exists a multicast transmission scheme over a large enough finite field $\mathbb{F}_q$, in which intermediate network nodes linearly combine their incoming information symbols over $\mathbb{F}_q$, that delivers the information from the sources simultaneously to each receiver at a rate equal to h.

Even in very simple network topologies, choosing a too small field size can reduce the effectiveness of the coding by introducing an excess of linearly dependent packets [5]. However, note that in RLNC schemes increasing the field size also increases the overhead added to each encoded symbol through the encoding vector. It is therefore undesirable to increase the field size more than necessary, an overview of this trade-off is provided in [6].

B. Memory Consumption

On memory constrained devices, algorithms with a limited memory consumption may be a requirement. Many finite field implementations used in NC applications rely on different variants of lookup tables to provide efficient arithmetic operations. On constrained devices lookup tables may be undesirable as they occupy valuable space in memory (e.g. on a sensor board) and can cause severe performance issues due to the typically limited Central Processing Unit (CPU) cache size [3].

In this paper we will introduce the initial implementation results utilizing a new scheme based on results from Optimal Extension Fields (OEFs) and a sub-field mapping algorithm developed by Crowley et al. [7]. This scheme allows for both a high field size and a very low memory consumption.

The remainder of the paper is organized as follows. In Section II OEFs are described. Section III introduces the algorithm for fast modulo reduction. Section IV introduces the binary sub-field mapping algorithm. Section VII presents the obtained results from an initial implementation. The final conclusions are drawn in Section VIII.

II. Optimal Extension Fields

OEFs were introduced in [8] for use in public-key cryptographic systems. OEFs differ from the traditional binary field implementations by changing the characteristic of the field. The general definition of a field is given as $\mathbb{F}_{p^m}$, for which p is a prime number also called the field characteristic and m denotes the extension used. Efficient finite field arithmetics

in an OEF are achieved by choosing a characteristic (i.e. a prime p) close to the word size of the underlying hardware processor e.g. 8, 16, 32 or 64 bits and creating field extensions using irreducible polynomials of a special form. This approach differs from traditional implementations where a binary extension field is used, i.e. fields with characteristic $p = 2$. As an example, a concrete implementation for a 32 bit CPU may use a binary extension field such as $\mathbb{F}_{2^{32}}$, whereas an OEF implementation could use $\mathbb{F}_{4294967291^m}$, where 4294967291 is the prime number $2^{32} - 5$. Although Bailey et al. envisioned OEFs to be used in cryptographic systems, we may use their results to also create efficient prime field implementations for NC applications.

Before moving on, let us give the definitions describing an OEF as described in [8]:

- **Definition 1.** A pseudo-Mersenne prime is a prime number of the form $2^n - c$, where $\log_2(c) \leq \frac{1}{2}n$
- **Definition 2.** An Optimal Extension Field is a finite field $\mathbb{F}_{p^m}$ such that:
  1) p is a pseudo-Mersenne prime.
  2) An irreducible bionomial $p(x) = x^m - \omega$ exists over $\mathbb{F}_p$.

  We observe that there are two special cases of OEF which yield additional arithmetic advantages, which we will call Type I and Type II.
- **Definition 3.** A Type I OEF has $p = 2^n - 1$. A Type I OEF allows for subfield modular reduction with very low complexity as we will show later.
- **Definition 4.** A Type II OEF has an irreducible binomial $x^m - 2$. A Type II OEF allows speedups in extension field modular reduction.

In the following we will show how the ideas of OEFs can be utilized in a NC context. Since most NC systems do not require the same large fields as certain cryptographic systems, we will not discuss the construction of extension fields, but focus on the OEF sub-field which is given as a regular prime field $\mathbb{F}_p$, where p is a pseudo-Mersenne prime. For this reason we will refer to this type of finite field as an Optimal Prime Field (OPF). To implement this in practice, two open questions must be addressed: How to implement fast modulo reduction in the chosen prime field $p = 2^n - c$ and how to ensure that arbitrary input data can be represented within the chosen field.

III. Fast Sub-Field Modulo Reduction

This result originally stems from [9] and was later utilized in OEFs. In a finite field, the modulo operation is used to ensure that all arithmetic operations performed in the field remain "closed", i.e. adding, subtracting, multiplying or dividing two field elements must result in an element also in the field e.g. in $\mathbb{F}_p$, where $p = 7$:

$$(5 + 4) \mod 7 = 2 \tag{1}$$
$$(5 - 4) \mod 7 = 1 \tag{2}$$
$$(5 \cdot 4) \mod 7 = 6 \tag{3}$$
$$(5/4) \mod 7 = 3 \tag{4}$$

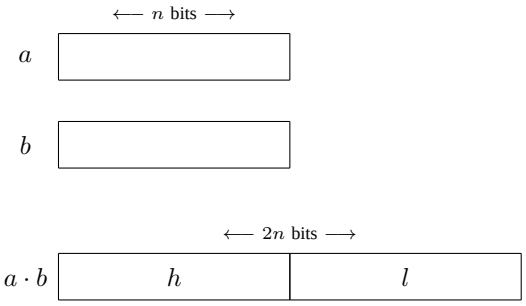
To create an efficient prime field implementation, an efficient way of implementing the modulo reduction is therefore also needed. Given two n-bit integers a, $b \in \mathbb{F}_p$ we may handle the addition and subtraction with reduction modulo p as:

$$a + b = \begin{cases} a + b & \text{if } a + b < p \\ a + b - p & \text{if } a + b \geq p \end{cases} \tag{5}$$

$$a - b = \begin{cases} a - b & \text{if } a - b \geq 0 \\ a - b + p & \text{if } a - b < 0 \end{cases} \tag{6}$$

For multiplication and division the process is more involved. Division can be considered a multiplication with the inverse element, therefore it will only be necessary to perform the modulo reduction after multiplication. The inverse of a field element can be calculated using the extended Euclidean Algorithm, see [10]. As with addition and subtraction, we again consider two n-bit integers a, $b \in \mathbb{F}_p$. Recall from "Definition 1" that p is a pseudo-Mersenne prime of the form $p = 2^n - c$, where $\log_2(c) \leq \frac{1}{2}n$. The goal is to calculate the modulo reduction of the product of two n-bit integers using only additions, shifts and multiplications. Although this may sound more complicated than calculating the modulo reduction using an integer division it will often be faster in practice due to the high latency of the integer division instruction on the CPU [11]. In the following the idea behind the modulo reduction algorithm is presented.

First choose a pseudo-Mersenne prime of the form $p = 2^n - c$, where $\log_2(c) \leq \frac{1}{2}n$. Calculate the product of the two n-bit integers a and b:



The result $a \cdot b$ can be represented as a n-bit high-half h and a n-bit low-half l. From this, it can be seen that the product $a \cdot b$ can be rewritten as:

$$a \cdot b = h \cdot 2^n + l \mod p \tag{7}$$

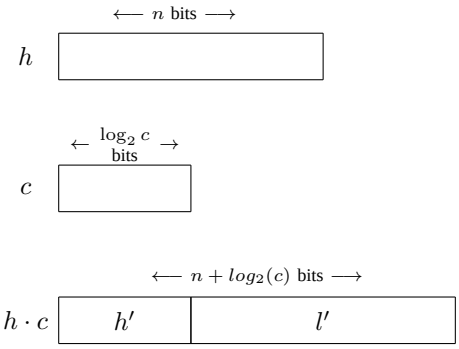
Rewriting the product as a sum of the low- and high-half, we see that the factor 2^n can be reduced using the chosen prime p, where $p = 2^n - c$:

$$2^n \equiv c \mod 2^n - c \tag{8}$$

Substituting this result into Equation (7) yields:

$$a \cdot b = h \cdot c + l \mod p \tag{9}$$

Graphically we may represent the product $h \cdot c$ as:



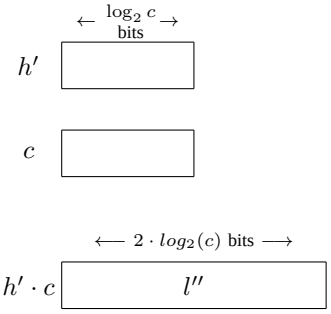
Again we represent the result as a high-half added to the low-half:

$$h \cdot c = h' \cdot 2^n + l' \mod p \tag{10}$$

Repeating the reduction step using Equation (8) we may rewrite the product:

$$h \cdot c = h' \cdot c + l' \mod p \tag{11}$$

The product $h' \cdot c$ can now be represented as:



Where l'' can be represented using $2 \cdot log_2(c) \leq n$ bits. This is possible since we have chosen c as $\log_2(c) \leq \frac{1}{2}n$. This represents the final step in the algorithm as no further high-half bits are produced. We may note that due to our choice of c this is guaranteed to happen. By combining the results from Equation (9) and (11), we may rewrite the multiplication as a sum of three $\leq n$ bit integers:

$$a \cdot b = l + l' + l'' \mod p \tag{12}$$

As shown for addition in Equation (5), we only have to ensure that the sum is below p by subtracting p if necessary. This leads to the following algorithm for multiplication:

Listing 1: Optimal prime field multiplication for p

```
def optimal_multiply(a, b):
    """
    Computes and returns the product a*b modulo
        the prime p.

    a, first integer operand
    b, second integer operand
    """
    c = 5
    p = 2**32 - c

    x = a*b
    while x >= 2**32:
        high = x >> 32;
        low = x & 0xffffffff;

        x = high * c + low;
    if x >= p:
        x = x - p;

    return x
```

1) Example: The following example shows the how the algorithm computes the modulo remainder for two large 32 bit integers.

Initialization:

$$x = 25351674994116735234$$
$$a = 5294969294 \qquad b = 4787879511$$
$$c = 5 \qquad p = 4294967291$$
$$high = 0 \qquad low = 0$$

Iteration 1:

$$x = 32021112688$$
$$a = 5294969294 \qquad b = 4787879511$$
$$c = 5 \qquad p = 4294967291$$
$$high = 5902646806 \qquad low = 2507878658$$

Iteration 2:

$$x = 1956341651$$
$$a = 5294969294 \qquad b = 4787879511$$
$$c = 5 \qquad p = 4294967291$$
$$high = 7 \qquad low = 1956341616$$

In practice this algorithm can be implemented in an unrolled fashion (without the while loop).

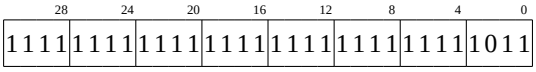
IV. Sub-Field Data Mapping

As mentioned, one of the goals of OEFs is to match the underlying processor word size as closely as possible. However, in practice this means that the finite field elements cannot be exactly represented by any common data types i.e. 8, 16, 32 or 64 bits, due to the fact that a prime must be used. As an example, using a prime field with characteristic

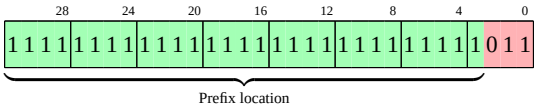
$p = 2^{32} - 5$, we see that it does not allow the binary values from `0xfffffffb` to `0xffffffff`. This would create problems for most NC applications, where the data being processed cannot be guaranteed to be within a certain binary range. Consequently, in this example we require an encoding/decoding scheme for mapping arbitrary 32-bit data values into values in the $[0, 2^{32} - 5)$ range. Using the approach presented by Crowley et al. [7], this goal may be achieved with limited computational overhead. The algorithm may be summarized in the following steps:

1) Choose a data type of n bits to match the processor word size.
2) Select a pseudo-Mersenne prime of the form $p = 2^n - c$.
3) Partition the input data into blocks of maximum $2^t - 1$ data words, where $t \leq \lceil n - log_2(c) \rceil$.
4) For each block, search the data to find a t-bit prefix not present in the data. Note, that since we have a maximum block size of $2^t - 1$, this prefix is guaranteed to exist.
5) Negate the t bit prefix and XOR it with all n-bit words in the data block. This will ensure that all data values are representable in the chosen prime field.
6) When no more finite field operations are required, e.g. after the data block has been transmitted and decoded, reverse the prefix mapping by again performing the XOR with the negated prefix on the data.

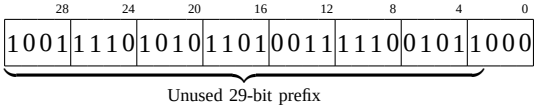
The outlined algorithm will ensure that all values larger than the selected prime will be mapped to a value representable with in the prime field. As an example of how the binary mapping works, consider the binary representation of the pseudo-Mersenne prime $p = 2^{32} - 5$:



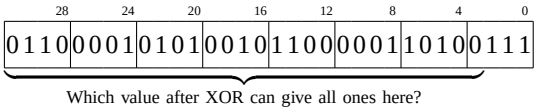
One way to make sure that all input values are below the prime is to ensure that at least a single zero will appear in the top 29 bits, denoted as the *prefix*:



The binary sub-field mapping algorithm achieves this by partitioning the input data into blocks with a maximum of $2^{29} - 1$ data words. Corresponding to a maximum block size of approximately 2.1 GB. This block size guarantees that a 29 bit prefix, s, may be found which does not appear in any of the data words. Using the prefix s, a binary mask can be constructed and XOR'ed with the remaining data block values. The binary mask is simply the negated version of the prefix. To illustrate how this works, assume we have found a prefix which does not appear anywhere in our data block:



Negate this prefix:



Only the unused 29-bit prefix.

Now notice that the only value for which the XOR with the negated prefix can produce all binary ones is the prefix itself - which does not exist in the data block. XOR'ing with all other input values with the negated prefix is guaranteed to have at least a single zero bit in the top t bits. Therefore, all data values in the block will be representable within the prime field.

In general the length of a data block will determine the number of bits necessary to ensure that an unused prefix can be found. This relationship between the block length b in 32-bit data words and prefix length t in bits is given by Equation 13.

$$t = \lceil \log_2(b) \rceil \text{ where } t \leq \lceil n - log_2(c) \rceil \tag{13}$$

As an example for a block length of 15 data words it is given that at least one 4-bit prefix must exist, which does not appear anywhere in the data.

V. Prefix Search

One drawback of this solution is that finding the prefix will require processing of the entire data block. This is however only necessary once, and may therefore be precomputed and attached to a storage object as meta-data. However, for steaming applications data is produced on-the-fly and the prefix search therefore has to be executed as the blocks are made available. In the following we will therefore investigate the added processing overhead of the prefix search. Searching for the prefix may be accomplished in several ways in the following we investigate the performance of two algorithms 1) *bitmap lookup* and 2) *k-pass binary search*.

As shown in Figure 1 the bitmap algorithm works by mapping every t-bit prefix pattern in a data block to an unique bit position in a bitmap with 2^t entries. Any unused prefix can therefore be found after a single iteration of the data block. As the number of possible prefix values grows exponentially with the length of data block this algorithm is expected to perform best if the block length remains small. The storage requirements of the bitmap algorithm is given in Equation 14.

$$bitmap_{memory} = \left\lceil \frac{2^t}{8} \right\rceil \quad [B] \tag{14}$$

The k-pass binary search performs k iterations over the data block to find an unused prefix. Utilizing more than a single iteration reduces the memory consumption but is expected to increase the algorithms computational cost. The general principle of the binary search algorithm is that given a t bit prefix we may inspect only a subset j of the bits, where $j < t$ to determine first j-bits of the unused prefix. This can be achieved by noticing that for every j-bit prefix there can be at most 2^{t-j} bit patterns. It is given that if a counter contains less than 2^{t-j} values there must exist an t bit prefix with

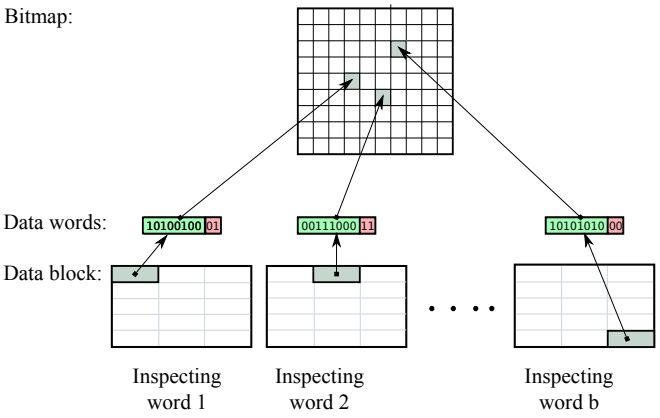


Fig. 1: Example of the bitmap algorithms. The algorithms iterates over the data block and maps each t-bit prefix to a corresponding bit-position in the bitmap. After iterating over all words in a data block the unused prefix can be found by iterating over the bitmap to locate an unused bit position.

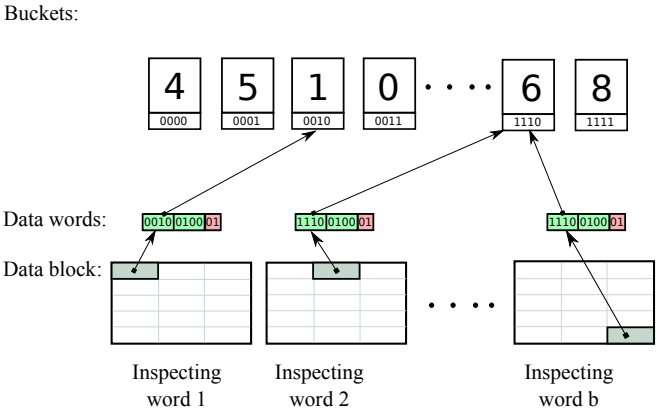


Fig. 2: Example of the binary search algorithm. The algorithm uses k passes over the data block. In each step it counts the number of occurences of j-bit prefixes.

the top j bits which does not appear in the data. Recall this is guaranteed to happen since we at most allow $2^t - 1$ data words in block. Equation 15 gives the relationship between k, t and j.

$$j = \left\lceil \frac{t}{k} \right\rceil \quad [bits] \tag{15}$$

As shown in Figure 2 the binary search algorithm counts all j-bit prefixes once. Utilizing the counters the algorithm choose a j-bit prefix for which it knows that an unused t-bit prefix exists. Using the first j-bit prefix as a filter it continues by counting the next j-bit prefixes. This process continues until a counter contains the value 0, when this happens the concatenation of the chosen j bit prefixes will constitute the unused t-bit prefix. The value k determines the maximum number of iterations needed. Increasing k reduces the memory requirements as less counters have to be stored. The total memory requirements can be calculated as the number of counters needed, multiplied by the size of each counter (in our implementation we used `uint32_t` which is 4 bytes), as given in Equation 16.

$$binary\_search_{memory} = 2^j \cdot 4 \quad [B] \tag{16}$$

The storage requirements of both algorithms are shown in Figure 3. As shown it is possible to a large extend tune the memory consumption of the algorithms. Also worthwhile noticing is that even for a generation size of 2048 it is possible to run the prefix search using less than 1000 bytes of memory using the binary search $k = 4$.

A. Prefix Search Performance

In this section we benchmark the presented prefix search algorithms. The benchmark measures the time required for the algorithm to find a missing prefix in a block of random data. In

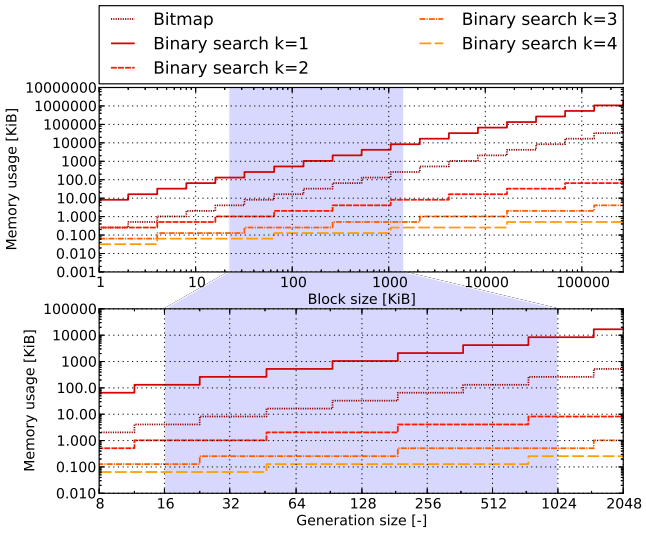


Fig. 3: The memory requirements given for each prefix search algorithm. Given both in terms of total block size (figure top) and in terms common choice of generation size (figure bottom). A packet size of 1400B was used to calculate the block size of the generations.

order to get an impression of the performance of the algorithms the benchmark have been conducted on the following device:

TABLE I: Specifications of the device used for benchmarking

Device	Desktop PC
OS	XUbuntu 12.04
CPU	Intel(R) Core(TM) i7 920 @ 2.67 GHz
Cache	L1 128 KiB, L2 1 MiB, L3 8 MiB
Memory	6 GB DDR3 1066 MHz

In Figure 4 we see the results from the benchmarks. The measurements show that the fastest algorithm for all block sizes is the Binary search with $k = 2$. Which is able to find the

missing prefix in approximately 6 ms for generation size 1024. For generation sizes below 256 which are quite common in NC applications, due to the otherwise high decoding complexity, the prefix search completes within 2 ms. Also we may notice the "staircase" like increase in search time as the block size increases. These slowdowns correspond to the block sizes where the algorithms require one more bit in the prefix. This has the effect that the amount of available prefixes double and therefore the algorithm have to perform more bookkeeping. As an example when the prefix length increases from 15 to 16 bits the Bitmap algorithm increases its storage requirements from 32 KB to 64 KB. Following an increase in the algorithms storage requirements the search time grows slowly as the block size increase, until the prefix needs additional bits causing a new performance slowdown. This effect is especially visible for large block sizes as shown in Figure 4 (top).

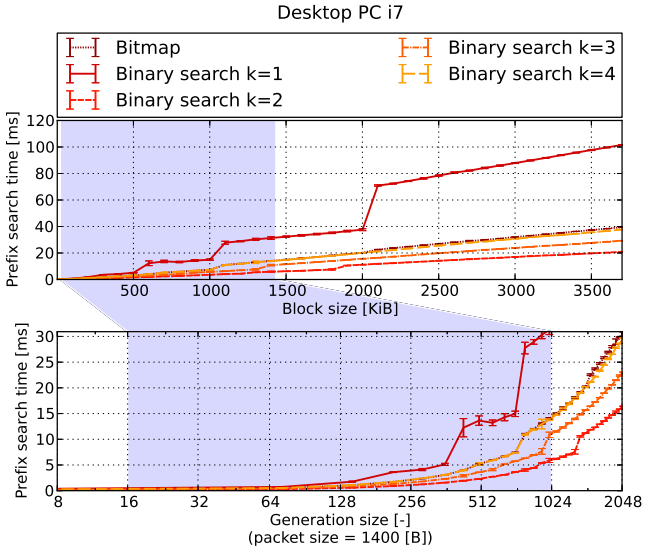


Fig. 4: Time in milliseconds required to search for the needed block prefix on the desktop i7. The block size was increased in steps of 10KiB for each step 100 runs where completed. The vertical bars show the standard deviation in each step.

In Table II we show the results for the fastest algorithm Binary search $k = 2$, these result signify the overhead which would be added per block due to the prefix search. Depending on the type of application this could be problematic, e.g. for live streaming applications the prefix must be found "on-the-fly" as data is being produced, whereas for static objects such as files the prefix may be found in advance and attached to the data as meta-information.

TABLE II: Prefix search time [ms] for Binary search $k = 2$

Generation size	16	32	64	128	256	512	1024
Time [ms]	0.19	0.23	0.27	0.54	1.09	2.30	5.99

VI. Network Coding Arithmetics

In order to quantify the potential performance of different finite field implementations it is necessary to understand the statistics of the arithmetic operations performed in NC algorithms. For a Gaussian Elimination decoder and a generation size g, packet size of f finite field elements and an encoding vector of v finite field elements we expect the upper-bound for the operations as shown in Table III.

TABLE III: Upper bound operations for a Gaussian Elimination decoder

Operation	`dest[i] = dest[i] - (constant * src[i])`
Upper-bound	$O((g^2 - g) \cdot (v + f))$
Operation	`dest[i] = dest[i] * constant`
Upper-bound	$O(g \cdot (v + f))$
Operation	`invert(value)`
Upper-bound	$O(g)$

Notice, that the length of the encoding vector is always $v = g$, however to make it clearer where the operations are coming from we will keep using v to denote its length. For the encoding algorithms we have the upper-bound expression shown in Table IV.

TABLE IV: Upper bound operations for a standard RLNC encoder

Operation	`dest[i] = dest[i] + (constant * src[i])`
Upper-bound	$O(g^2 \cdot f)$

To confirm these expressions the following results were obtained by instrumenting one standard RLNC encoding algorithm and a Gaussian Elimination based RLNC decoding algorithm (source code available here [12]). Field coefficients used for the encoding where drawn uniformly. It should also be noted that the algorithms require two implementations one for the binary field and one for higher order fields i.e. binary extension fields as well as the optimal prime field. This is required since high field size implementations do multiplication with non-trivial constants, whereas the binary field only uses addition and subtraction. This difference can be clearly seen in Figure 6. Whereas the higher order fields shown in Figure 5 heavily relies on multiplication.

These results confirm the expectation from the upper-bound expressions given in Table III and IV. Based on this can also be seen that in order to improve the performance of binary field algorithms the addition and subtraction should be optimizes. Whereas for higher order fields the compound operations addition and subtraction with a constant multiplication is the most common operation.

VII. Finite Field Arithmetics

This section presents the benchmark results of the suggested finite field algorithm.

Based on the results from the previous section the majority of operations performed are the two compound operations:

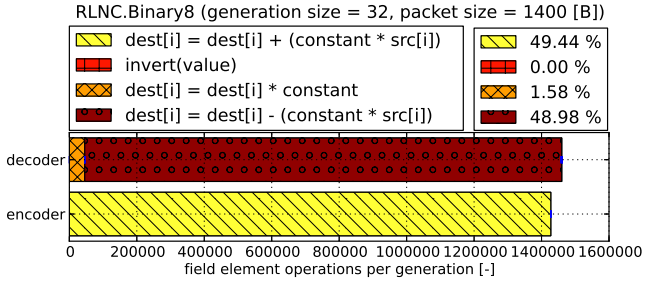


Fig. 5: Number of finite field operations needed during encoding and decoding of a single generation. The field used corresponds to the binary extension field $\mathbb{F}_{2^8}$, which means that every field element corresponds to 1 byte.

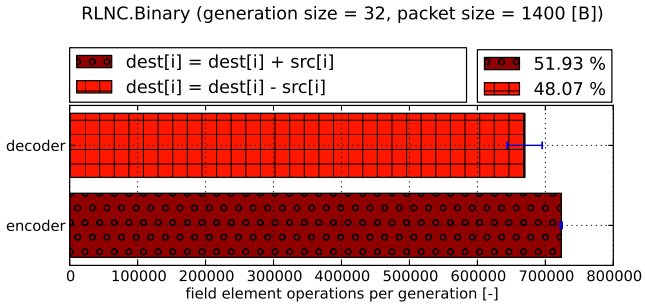


Fig. 6: Number of finite field operations needed during encoding and decoding of a single generation using the binary field $\mathbb{F}_2$. In this field every field element corresponds to 1 bit. Since operating on single bit values is very inefficient implementations typically operate on vectors of 1 bit elements. In the used implementation the data type used corresponds to a 1 byte i.e. 8 bit elements. The result shown above should therefore be scaled by 8 to get the 1 bit operations performed.

`dest[i] = dest[i] + (constant * src[i])`and `dest[i] = dest[i] - (constant * src[i])`. These operations are therefore of key interest when comparing different finite field implementations for NC algorithms. In the following we have tested a number of different finite field implementations (source code available here [13]):

- *SimpleOnline*$\{8, 16\}$: This algorithm computes the result on-the-fly in $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$ using an iterative algorithm, without any precomputed lookup table.
- *OptimalPrime2325*: This corresponds to the algorithm presented in this paper. Using the prime field $\mathbb{F}_{4294967291}$, where $p = 2^{32} - 5 = 4294967291$
- *FullTable8*: This algorithm utilizes a fully precomputed lookup table stored in memory to calculate the results in $\mathbb{F}_{2^8}$.
- *LogTable*$\{8, 16\}$: This algorithm uses a reduced lookup table to calculate the results in $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$. The log table minimize memory consumption at the cost of additional operations for every calculation.
- *ExtendedLogTable*$\{8, 16\}$: This algorithm extends the lookup table used by the *LogTable* to calculate the results in $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$. The extended lookup table removes a number of checks necessary in the *LogTable* algorithm when moving from exponential to polynomial representation.

The following figures show the throughput for the arithmetic operations tested. The benchmark uses two generations containing each g packets where each packet is 1400 B long. From each generation two packets are then randomly selected and the specified operation is performed. In the operations tested a constant is used for the multiplication, this constant was randomly generated for each invocation of the operations under test. For each operation number of iterations were completed so that the total measurement time exceeded a minimum of 10 ms, this was done to keep inaccuracies due to timing granularity and other disturbances in the measurements low. For each operation this was repeated 100 times.

The benchmarks were run on the device specified in Table I using three different generations sizes; $g = 32$, $g = 128$ and $g = 1024$. These numbers were chosen to see how the algorithms were affected by the working set size (which is the generation size multiplied by the packet size). The working set size can have a significant impact on performance due to caching effects [14, p. 593-673]. In this case we only observe a slight drop in performance as the working set size increases indicating that the CPU is able to keep the working set in the cache.

All implementations presented here are written in C++ using no assembler or compiler intrinsics to further speed up the computations. However, inspecting the assembly output of the compiled benchmark does reveal that the compiler was able to take advantage of vectorized Single Instruction Multiple Data (SIMD) instructions in some of the arithmetic loops. Although this boosts performance considerable some functions were not optimized by the compiler. It is therefore likely that further performance could be achieved by hand-writing some of operations using assembly or vectorized SIMD instructions directly.

In Figure 7 we see the benchmarks for a generation size of 32. In this case we see that the OPF preforms better than the alternative implementations. One interesting thing to observe is that the OPF performs slightly faster in subtraction than addition. The explanation for this is the that addition requires two checks, one for checking for integer overflow, and one for checking whether the prime modulo operation must be performed. For subtraction we only have to check for integer underflow. Avoiding this additional check yields an approx. 5% performance increase.

In Figure 8 we see the benchmarks for a generation size of 128 and finally in Figure 9 we see the benchmarks for a generation size of 1024. Where the tendency remains the same as for the generation size of 32.

As seen the two implementations `FullTable8` and `OptimalPrime2325` are by far the fastest. Where the `OptimalPrime2325` on average provides an 18% performance increase for addition and an average 23% performance

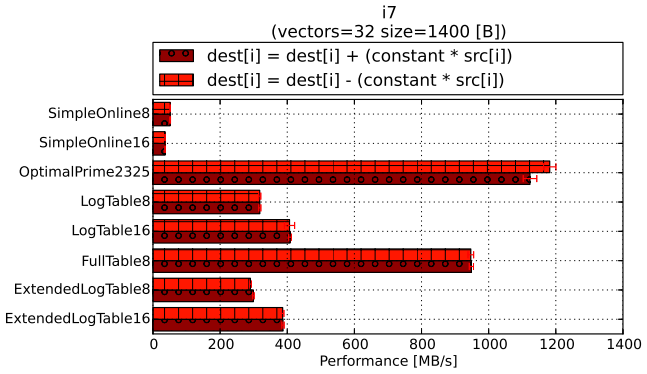


Fig. 7: Throughput for the compound NC operations for a generation size of 32.

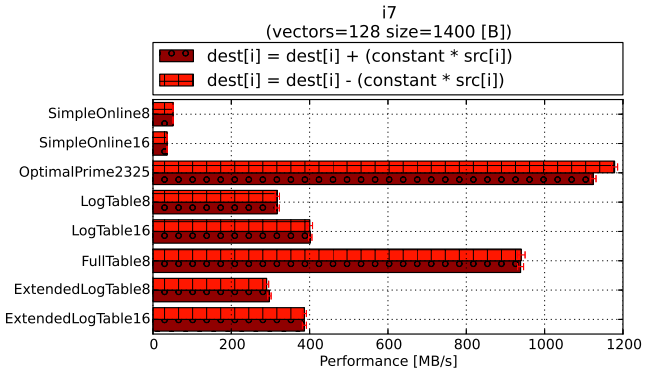


Fig. 8: Throughput for the compound NC operations for a generation size of 128.

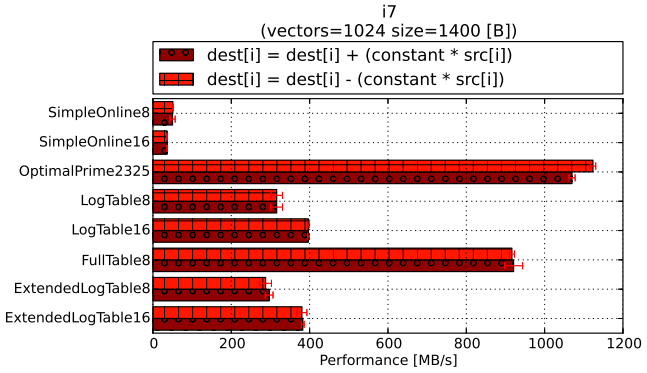


Fig. 9: Throughput for the compound NC operations for a generation size of 1024.

increase for subtraction when compared to the `FullTable8`. Table V shows the measured results for the two benchmarks.

TABLE V: Throughput measurements in MB/s for the `OptimalPrime2325` and `FullTable8` algorithms.

generation size	algorithm	operation	throughput [MB/s]
32	`OptimalPrime2325`	add	1122.44
		subtract	1181.02
	`FullTable8`	add	948.14
		subtract	946.14
128	`OptimalPrime2325`	add	1124.21
		subtract	1178.35
	`FullTable8`	add	938.15
		subtract	939.78
1024	`OptimalPrime2325`	add	1069.69
		subtract	1122.75
	`FullTable8`	add	920.80
		subtract	916.31

VIII. Conclusion

In this paper we have presented our initial investigation regarding the use of OPF for NC applications. The results show that OPF looks like a promising addition to the selection of finite field implementations. Besides the good performance one of the main benefits from the OPFs is the large field size and the limited memory consumption required by the algorithms. As briefly mentioned the implementation presented here uses a plain C++ implementation, and inspecting the compiled assembly we could verify that the compiler did not optimize all operations using SIMD instructions. It is therefore likely that such an implementation could yield even faster implementations. The main drawback of this approach is the need for the prefix binary mapping scheme. Further work should be invested in reducing the overhead added by this. We expect OPFs to be particular useful on very small embedded devices, with only KB's of memory, since these devices cannot use the lookup table based algorithms.

Acknowledgments

This work was partially financed by the CONE project (Grant No. 09-066549/FTP) granted by Danish Ministry of Science, Technology and Innovation as well as by the collaboration with Renesas Mobile throughout the NOCE project.

References

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

[2] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[3] K. M. Greenan, E. L. Miller, and T. J. E. Schwarz, "Optimizing galois field arithmetic for diverse processor architectures and applications," in *MASCOTS*, E. L. Miller and C. L. Williamson, Eds. IEEE Computer Society, 2008, pp. 257–266.

[4] C. Fragouli and E. Soljanin, "Network Coding Fundamentals," *Foundations and Trends in Networking*, vol. Vol. 2, Issue 1, pp. 1–133, 2007.

[5] M. V. Pedersen, J. Heide, F. Fitzek, and T. Larsen, “Network coding for mobile devices - systematic binary random rateless codes,” in *Workshop on Cooperative Mobile Networks 2009 - ICC09*. IEEE, Jun. 2009.

[6] J. Heide, M. V. Pedersen, F. H. Fitzek, and M. Médard, “On code parameters and coding vector representation for practical rlnc,” in *IEEE International Conference on Communications (ICC) - Communication Theory Symposium*, Kyoto, Japan, 5-9 June 2011.

[7] P. Crowley. (2006, Nov.) Gf(232-5). [Online]. Available: http://www.lshift.net/blog/2006/11/29/gf232-5

[8] D. V. Bailey and C. Paar, “Optimal extension fields for fast arithmetic in public-key algorithms,” in *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1998, pp. 472–485. [Online]. Available: http://portal.acm.org/citation.cfm?id=646763.706317

[9] S. B. Mohan and B. S. Adiga, “Fast algorithms for implementing rsa public key cryptosystem,” *Electronics Letters*, vol. 21, 1985.

[10] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[11] T. Granlund. (2011, Mar.) Instruction latencies and throughput for amd and intel x86 processors. [Online]. Available: http://gmplib.org/tege/x86-timing.pdf

[12] Steinwurf ApS. (2012) Kodo git repository on github. Http://github.com/steinwurf/kodo.

[13] ——. (2012) Fifi git repository on github. Http://github.com/steinwurf/fifi.

[14] R. E. Bryant and D. R. O’Hallaron, *Computer Systems: A Programmer’s Perspective*, 2nd ed. USA: Addison-Wesley Publishing Company, 2010.

Paper 6

A Mobile Application Prototype using Network Coding

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

European Transactions on Telecommunications, Vol. 21, No. 8, 12.2010, p. 738-749.

EUROPEAN TRANSACTIONS ON TELECOMMUNICATIONS
Eur. Trans. Telecomms. 2010; **21**:738–749
Published online 11 October 2010 in Wiley Online Library
(wileyonlinelibrary.com) DOI: 10.1002/ett.1448

A mobile application prototype using network coding†

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

Department of Electronic Systems, Aalborg University, Denmark

SUMMARY

This paper looks into implementation details of network coding for a mobile application running on commercial mobile phones. We describe the necessary coding operations and algorithms that implements them. The coding algorithms forms the basis for a implementation in C++ and Symbian C++. We report on practical measurement results of coding throughput and energy consumption for a single-source multiple-sinks network, with and without recoding at the sinks. These results confirm that network coding is practical even on computationally weak platforms, and that network coding potentially can be used to reduce energy consumption. Copyright © 2010 John Wiley & Sons, Ltd.

1. INTRODUCTION

Network Coding (NC) has received a lot of attention since the term was coined by Ahlswede *et al*. [1]. Several research works have investigated [2, 3] and implemented [4, 5] NC to prove the feasibility of this novel technique. NC can be applied in many communication scenarios such as multicast or meshed networking, where NC delivers promising results for throughput and reliability. While most codes are end-to-end, with NC packets can be recoded at each node in the network, which can be of special interest in multi-hop networks.

The concept of NC has been proven to work in theory, some current questions are how to design NC algorithms and whether these algorithms are too complex for a given platform. In References [6, 7], it has been shown that NC can be applied to sensor networks and meshed networks formed by mobile phones. One finding was that NC techniques must be designed with care if they are to be applied to the mobile or embedded domain. These platforms have limited resources such as energy, memory and computational power in addition to the general problems in mobile networking such as limited wireless capacity.

This paper introduces a mobile demo application using NC that is running on the Symbian/S60 platform used on most Nokia smartphones and by other manufactures such as Motorola, Samsung and Sony Ericsson. The main idea is that users wish to share content over short range wireless technologies such as WiFi. Instead of uploading the content to social networks such as MySpace or Facebook, the content can be conveyed directly to nearby mobile phones, which would allow a user to easily share photos with his/her friends ad hoc.

The use of NC is motivated by the fact that the transmission from one source to many sinks must be done in a reliable and efficient manner. NC enables this as it allows for efficient spectrum usage and a low complexity error control system. NC can be applied at different protocol layers, ranging from the physical layer over the network layer to the application layer. In this work we focus on the application layer. Furthermore, the paper provides some implementation guidance on how to keep the complexity of NC low.

* Correspondence to: Morten V. Pedersen, Department of Electronic Systems, Aalborg University, Denmark. E-mail: mvp@es.aau.dk

† A previous version of this paper was presented in the 15th European Wireless Conference (EW 2009), Aalborg, Denmark.

Received 30 November 2009
Revised 12 July 2010
Accepted 14 July 2010

The remainder of this work is organised in the following sections. Section 2 introduces different transmission approaches. Section 3 introduces NC operations and algorithms. In Section 4 the functionality and interface of the application prototype is introduced. Section 5 presents the obtained results. Section 6 provides a discussion on important considerations when implementing NC. The conclusion is presented in Section 7.

2. TRANSMISSION APPROACHES

Different approaches for transmitting the data are possible, here we present some possibilities. We assume that a single source s broadcast data to N sinks $t_1 \ldots t_N$ and that the source has a direct wireless link to the sinks, as shown in Figure 1. The data can be divided into a number of packets, g. Transmitting packets over the wireless link may lead to packet loss due to the characteristics of the wireless channel thus an error control system is needed.

2.1. Unicast

The simplest solution is for the source to send the data in a round robin fashion using a reliable unicast protocol e.g. Transmission Control Protocol (TCP). Such an approach is fully reliable as each sink is served individually. Each sink acknowledges received packets and therefore the source device can determine when all sinks have received all packets. This solution is simple and the computational complexity is low. However, if N is high the amount of redundant information sent from the source becomes significant.

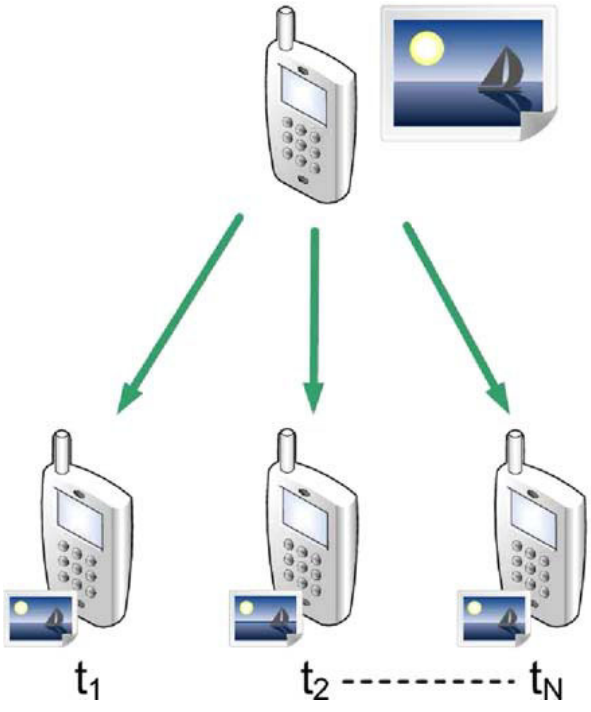


Figure 1. The basic PictureViewer setup.

2.2. Broadcast

Instead of sending to each device individually the source could broadcast the data to the sinks. This approach is highly efficient as long as no errors occur on the wireless link. However, when packet losses occur some form of error correction is needed. To achieve reliability the source needs to know which packets have been lost by one or more sinks and those must be retransmitted, this introduces the need for feedback information which consumes spectrum and time. The amount of feedback information depends on N and the Packet Error Probability (PEP). The feedback messages can be fairly small and as such do not require a lot of spectrum. However, they potentially introduces collisions in the network as both the source and sinks will attempt to transmit packets simultaneously. Thus the performance of such a broadcast approach depends on the effectiveness of the Medium Access Control (MAC).

Furthermore, the retransmissions by themselves is suboptimal as not all sinks will lose the same packets, thus each retransmitted packet will only be useful for a subset of the sinks. For example, if mobile devices 1, 2 and 3 have lost packet 17, 21 and 16, respectively, three broadcast packets must be transmitted, and each retransmitted packet is only useful for a single sink. Generally broadcast can be faster than unicast if $N > 1$ and its performance is less sensitive towards the number of sinks.

2.3. Pure network coding

One NC approach that lends itself to this scenario, is Random Linear Network Coding (RLNC) [2]. With this approach coding is used to simplify the problem of correcting lost packets at the sinks and furthermore reduces the requirement for feedback. In NC, nodes can combine the information in the network to create new packets [8]. Hence, the source codes $g + r$ packets from the g original packets and broadcasts these packets. r is the number of redundant packets and should be chosen according to the PEP of the link. Each sink only has to receive any g linear independent packets, which can then be decoded to recreate the original packets.

DOI: 10.1002/ett

Table I. Estimates of the achievable capacity, C, decoding delay, D, computational complexity, O, and energy consumption, E, when $N >> 1$.

	C	D	O	E
Unicast	Low	Low	Low	High
Broadcast	Med.	Low	Low	Med.
Pure NC	High	Med.	High	Med.
Systematic NC	High	Med.	Med.	Low

The advantage of NC can be illustrated by the previous example. In this case the source could code packets 16, 17 and 21 together into a new packet of the same length as the original packets. This packet is broadcasted to the three sinks, which each remove from the coded packet the packets they already got and thus decode the packet into the packet they lost. Thus, the retransmission that needed three transmissions using broadcast can be done by a single transmission using NC.

As the coding and decoding operations introduces complexity the computational requirement is increased. These operations will increase the Central Processing Unit (CPU) load and thus the energy consumption. However, the number of redundant packets transmitted from the source and feedback messages sent from the sinks can be decreased which help to decrease energy consumption.

2.4. Systematic network coding

To decrease the complexity systematic NC can be used [9]. Systematic NC combines the broadcast and NC approaches. As there is no obvious gain in coding the first g packets, the source broadcasts these packets and code the remaining r packets. Each uncoded packet is useful for all N sinks as they are linear independent. The following r packets are coded and have a high probability of being independent of the n uncoded packets. This approach decreases the computational complexity at the source and the sinks as only r packets has to be coded and decoded.

The different approaches are compared in Table I.

3. NETWORK CODING

This section introduces the coding operations necessary in NC and the algorithms used in the demo application, for details and analysis see Reference [10]. We base our solution on performing RLNC over a Galois field. When Galois fields are implemented on computer systems the Galois elements are generally of the form 2^i, where $i \in \mathbb{Z}^*$, and typically $i \in \{8, 16, 32\}$. We choose the smallest possible Galois Field, GF(2), to decrease the computational complexity of coding operations. This is done to overcome the challenges posed by the limited computational resources available on the test platform.

3.1. Coding operations

In NC data to be transferred from the source to the sinks is divided into packets of length m. The number of original packets over which encoding is performed is typically refereed to as the batch size or generation size and denoted g. Thus, the g original data packets of length m are arranged in the matrix $\boldsymbol{M} = [\boldsymbol{m}_1 \boldsymbol{m}_2 \dots \boldsymbol{m}_g]$, where $\boldsymbol{m}_i$ is a column vector.

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

Note that if an encoding vector consists of all zeros except a single scalar that is one, the coded packet is equal to an original packets and we say that it is trivially encoded.

3.1.2. Recoding. Any relay or sink node that have received $g - i > 1$ linear independent packets, can recode and thus create new coded packets. All received coded packets are placed in the matrix $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1 \hat{\boldsymbol{x}}_2 \dots \hat{\boldsymbol{x}}_{g-i}]$ and all encoding vectors are placed in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1 \hat{\boldsymbol{g}}_2 \dots \hat{\boldsymbol{g}}_{g-i}]$, we denote this the decoding matrix. The number of received linear independent packets $g - i$ is equal to the rank of $\hat{\boldsymbol{G}}$. $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{X}}$ is multiplied with a randomly generated vector $\boldsymbol{h}$ of length $g - i$, $\tilde{\boldsymbol{g}} = \hat{\boldsymbol{G}} \times \boldsymbol{h}$, $\tilde{\boldsymbol{x}} = \hat{\boldsymbol{X}} \times \boldsymbol{h}$. In this way we can construct $\tilde{\boldsymbol{G}} = [\tilde{\boldsymbol{g}}_1 \tilde{\boldsymbol{g}}_2 \dots \tilde{\boldsymbol{g}}_{g-i}]$ that contains $g - i$ randomly generated recoding vectors and $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1 \tilde{\boldsymbol{x}}_2 \dots \tilde{\boldsymbol{x}}_{g-i}]$ that consists of $g - i$ recoded data packets.

Note that $\boldsymbol{h}$ is only used locally and that there is no need to distinguish between coded and recoded packets when further recoding or decoding is performed.

3.1.3. Decoding. In order for a sink to successfully decode the original data packets, it must receive g linear independent coded packets and encoding vectors. All received coded packets are placed in the matrix $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1 \hat{\boldsymbol{x}}_2 \dots \hat{\boldsymbol{x}}_g]$ and all encoding vectors are placed in the matrix $\hat{\boldsymbol{G}} = [\hat{\boldsymbol{g}}_1 \hat{\boldsymbol{g}}_2 \dots \hat{\boldsymbol{g}}_g]$. The original data $\boldsymbol{M}$ can then be decoded as $\hat{\boldsymbol{M}} = \hat{\boldsymbol{X}} \times \hat{\boldsymbol{G}}^{-1}$.

DOI: 10.1002/ett

Note that the set of g linear independent packets can contain any mix of uncoded, coded and recoded packets.

3.2. *Coding algorithms*

In this section we present pseudo code for the coding operations in RLNC based on GF(2).

3.2.1. Encoding. A packet in GF(2) can be encoded in two simple steps. First the encoding vector, $\boldsymbol{g}$, of length g, is generated as a random bit vector, where the indices in the vector corresponds to packets in the original data set i.e. index one corresponds to packet one. The second step is performed by iterating over the encoding vector and adding packets where the corresponding index in the encoding vector is 1.

The following listing shows the encoding algorithm in pseudo code, where $\boldsymbol{M}$ is the data buffer containing all original packets, $\boldsymbol{g}$ is an encoding vector and $\boldsymbol{x}$ is the resulting encoded packet.

```
1:  procedure ENCODEPACKET (M,x,g)
2:      x = 0̄
3:      for each bit b in g do
4:          if b equal 1 then
5:              i = position of b in g
6:              x = XOR(x, M[i])
7:          end if
8:      end for
9:  end procedure
```

3.2.2. Recoding of the received $g - i$ packets in $\hat{\boldsymbol{M}}$ is performed similar to encoding. However, instead of combining all g original data packets, the received $g - i$ received packets are combined. First the recoding vector, $\boldsymbol{h}$, of length $g - i$, is generated as a random bit vector, where the indices in the vector corresponds to received packets i.e. index one corresponds to packet one. The second step is performed by iterating over the recoding vector and adding packets where the corresponding index in the encoding vector is 1. Simultaneously the packets corresponding encoding vectors are added in order to create a new encoding vector.

The following listing shows the recoding algorithm in pseudo code, where $\hat{\boldsymbol{M}}$ is the data buffer containing all received packets both partially and fully decoded, $\hat{\boldsymbol{G}}$ is the corresponding encoding vectors, $\boldsymbol{h}$ is the recoding vector, $\tilde{\boldsymbol{x}}$ is the resulting recoded packet, and $\tilde{\boldsymbol{g}}$ is the resulting encoding vector.

```
1:  procedure RECODEPACKET(M̂,Ĝ,h,x̃,g̃)
2:      x̃ = 0̄, g̃ = 0̄
3:      for each bit b in h do
4:          if b equal 1 then
5:              i = position of b in h
6:              x̃ = XOR(x̃, M̂[i])
7:              g̃ = XOR(g̃, M̂[i])
8:          end if
9:      end for
10: end procedure
```

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

In the second step we perform backward substitution with the newly identified pivot packet. This is done by subtracting the pivot packet from previously received packets for which the corresponding encoding vector indicates that the particular packet is a combination of the pivot packet.

The following listing shows the decoding algorithm in pseudo code, where $\hat{\boldsymbol{M}}$ is the packet decode buffer of packets received and decoded so far and $\hat{\boldsymbol{G}}$ is the corresponding encoding vector buffer, $\hat{\boldsymbol{x}}$ is a newly received encoded packet and $\hat{\boldsymbol{g}}$ is the newly received encoding vector.

```
1:  procedure DECODEPACKET(M̂,Ĝ,x̂,ĝ)
2:      pivotposition = 0
3:      pivotfound = false
4:      for each bit b in ĝ do          ( Forward substitution
5:          if b equal 1 then
6:              i = position of b in ĝ
7:              if i'th packet is in M̂ then
8:                  ĝ = XOR(ĝ,Ĝ[i])
9:                  x̂ = XOR(x̂,M̂[i])
10:             elseif pivotfound equal false
11:                 pivotfound = true
12:                 pivotposition = i
13:             end if
14:         end if
```


DOI: 10.1002/ett

```
15:     end for
16:     if pivotfound equal false then
17:         exit procedure                    ( Discard packet
18:     end if
19:     for each packet j in M̂ do ( Backward substitution
20:         k = Ĝ[j]
21:         if bit at pivotposition in k equal 1 then
22:             Ĝ[j] = XOR(Ĝ[j],ĝ)
23:             M̂[j] = XOR(M̂[j],x̂)
24:         end if
25:     end for
26:     Ĝ[pivotposition] = ĝ                  ( Insert packet
27:     M̂[pivotposition] = x̂
28: end procedure
```

The algorithm can also be used unmodified in a systematic coding approach, in which case we only have to ensure that uncoded packets are treated as pivot packets.

4. DEMO APPLICATION

A demo application, PictureViewer, has been developed to illustrate what is happening when NC is applied. Therefore, the PictureViewer application allows users to broadcast images located on their phones to a number of receiving devices. To illustrate the difference between different NC approaches the application allows users to monitor the decoding process directly. The decoding process is displayed by drawing the actual content of the decoding matrix onto the display of the receiving phones. As the application is primarily meant for demonstration purposes it may not be very useful in the real world. However, if the pictures where substituted with some other data, e.g. video or audio it might be useful for streaming in a local network or similar.

In Figure 2 the first column of screenshots shows the decoding process when pure NC is used. Here only coded packets are transmitted, and initially as shown in Figure 2(a) the content of the decoding matrix appears random. As the decoder receives more linear combinations, the decoding process solves the decoding matrix, and the original picture start to appear, see Figure 2(c). In Figure 2(e) the picture has been decoded and the transmission is complete. The second column of screenshots shows systematic NC, where all data is first transmitted uncoded. Figure 2(b) shows how uncoded packets are being inserted into the decoding matrix. In Figure 2(d) the application has entered the coding phase, where erasures which occurred during the uncoded phase are repaired by transmitting encoded packets. In this test the PEP was approximately 30% and therefore 70% of the data was received uncoded without need for additional decoding. This illustrates the advantage of the systematic approach as the number of packets that had to be decoded was reduced by 70%.

5. RESULTS

In this section we present the results of three measurements evaluating the performance of the used algorithms. The first tests focuses on the performance of the algorithms i.e. the amount of MB/s which can be encoded and decoded using the presented algorithms and the additional energy consumed. In the second test the code is used as an end-to-end code in an ad hoc Wireless Local Area Network (WLAN) to measure the impact of encoding and decoding. In the third test recoding is added and the sinks form a small cooperative cluster, hence the test provides information about the impact of recoding and simple cooperation. These test are intended to provide basic information about how the use NC impacts throughput and energy consumption, in a small ad hoc broadcast network comprising mobile devices with low computational capabilities.

5.1. Coding throughput

To determine the synthetic performance of the encoding and decoding algorithms we have implemented a coding library designed to deliver high throughput by optimising it through assembly and Single Instruction, Multiple Data (SIMD) instructions. This implementation was then ported to the Symbian platform and used in the PictureViewer application which allowed testing the algorithms on commercially available mobile phones. In the following tests the Nokia N95-8GB mobile phone with the following specifications was used; ARM 11 332 MHz CPU, 128 MB RAM, Symbian OS 9.2. In the throughput test a single phone was used to perform both the encoding and decoding operations by first encoding packets, saving the encoded data to memory, and subsequently decoding them. Packets were coded using the generation sizes $q = \{16, 32, 64, 128, 256\}$ and a packet size of 1200 bytes. This test was performed both for pure NC and systematic NC. For pure NC g coded packets were generated and subsequently decoded. To get an indication of the impact of using systematic NC a test was also be conducted where the first $0.7 \cdot g$ of the packets were uncoded and the last $0.3 \cdot g$ packets were coded. Thus, the coding performance corresponds to what would be expected if the packets were transmitted over a channel with PEP = 0.3.

DOI: 10.1002/ett

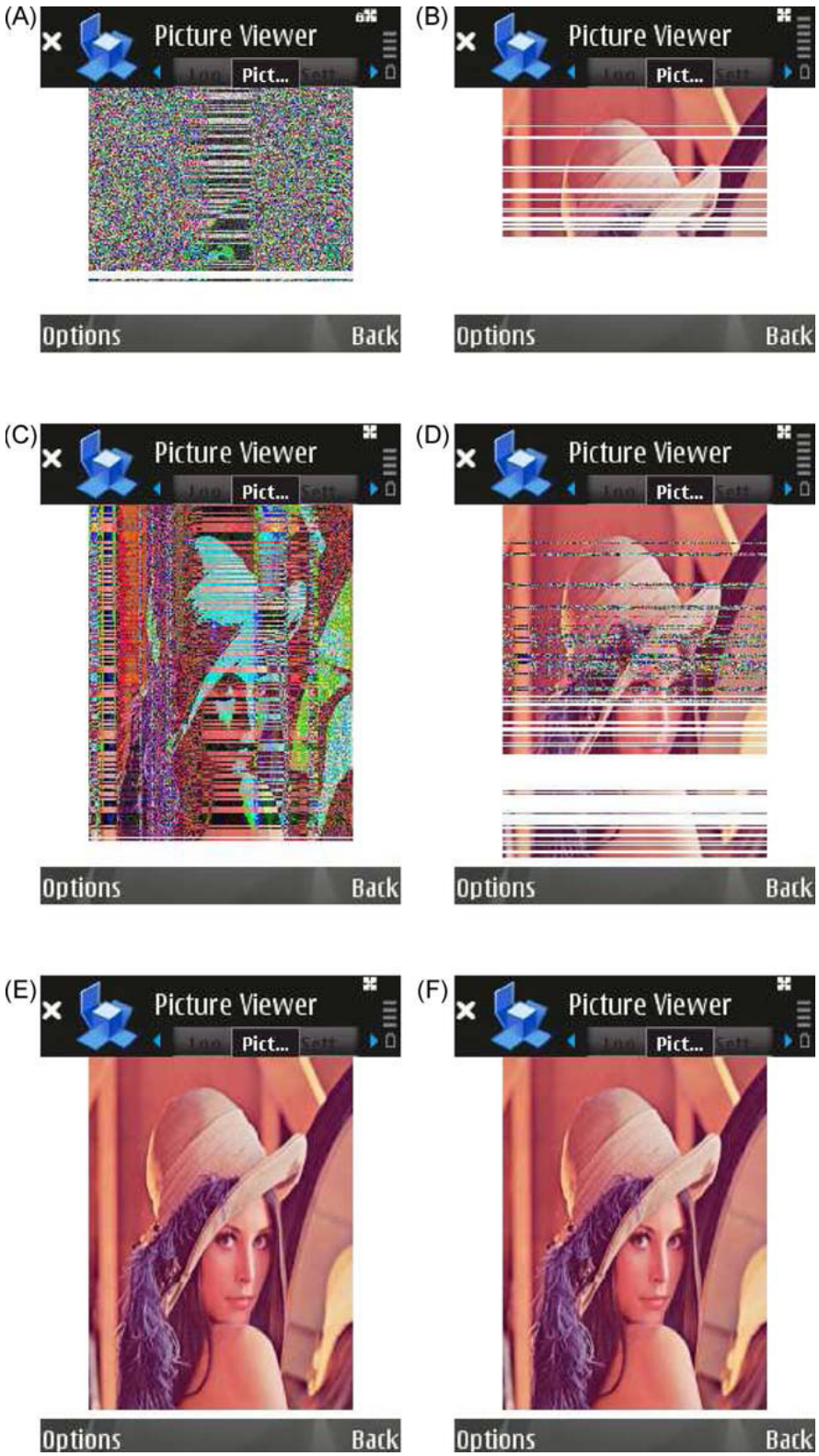


Figure 2. Pure NC: (a) partially decoded data, (c) image starting to appear as the decoders rank increases, (e) the final decoded image. Systematic NC: (b) received uncoded data, (d) erasures corrected by coded packets, (f) the final decoded image.

DOI: 10.1002/ett

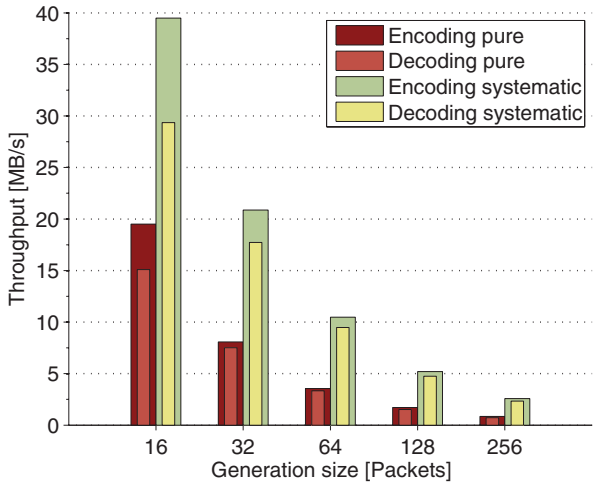


Figure 3. Synthetic throughput for encoding and decoding.

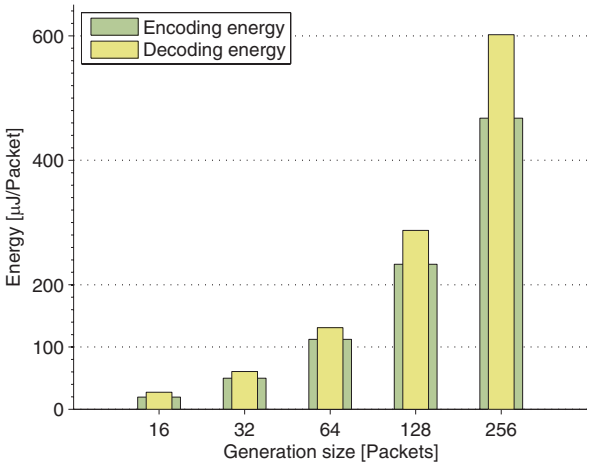


Figure 4. Energy spent per packet during encoding and decoding.

As seen in Figure 3 the encoding and decoding speed decreases as the generation size increases. Additionally the decoding throughput is somewhat lower than the encoding throughput. This is expected due to the higher computational complexity of the decoding algorithm. The test also shows that the systematic approach achieves approximately twice the throughput compare to the pure NC approach for a generation size of 16. For generation size of 64 and above the throughput is approximately tripled. Encoding and decoding of trivially coded packets require no computation, which results in a large speedup for the systematic approach. We note that the coding performance in a real network will depend on the ratio between uncoded and coded packets. In the extreme case where all packets are received coded, the two approaches perform identically and thus have the same throughput. The measured coding throughput indicate that the coding algorithms are fast enough to saturate the WLAN interface for all tested generation sizes, when compared to the achievable WLAN data rates of the Nokia N95 [11]. This result is important as the computational complexity introduced by coding should have minimal impact on the network and device performance, when compared to strategies without NC. A test was also performed to estimate the cost of coding in terms of energy. To measure the energy used for the coding operations, the Nokia Energy Profiler (NEP) was used during the tests. The results from the energy measurement are shown in Figure 4. To calculate the approximate energy consumption per coded packet, the test application first measured the idle power of the device using NEP. This was subtracted from the measured values during encoding and decoding, which gave the approximate power consumption caused by the coding operations.

As seen in Figure 4 the energy consumption per packet increases as the generation size grows. The power consumption was approximately constant during all tests, but as the throughput decreased the energy per packet increased. The energy used to decode a packet is slightly higher than for encoding, this can be attributed to the higher complexity of decoding and the following lower coding throughput.

5.2. *Coding in a network*

A simple approach to distribute the data from the source to all sinks in Figure 1 is to use systematic coding at the source and overshoot with a comfortable margin. Thus the source transmit so many packets that with a very high probability all sinks are able to decode, and as the source know nothing about the PEP for the sinks, r needs to be high. This is not very useful in a real network, but it allow us to observe the impact that the code has on the channel throughput and energy consumption, and thus choose parameters for the code that are appropriate for our test devices.

We measured from the first packet was received until the packet which completes the decoding of the generation was received. This allows us to obtain the performance as if we had a perfect feedback-channel and feedback scheme. The sink records the following parameters; time per generation, PEP, total packets, uncoded packets, coded packets and linear dependent packets.

The test was conducted using two Nokia N95s, one source and one sink. Packets were coded using the generation sizes $q = \{16, 32, 64, 128, 256\}$ and a packet size of 1200 bytes. Approximately 100.000 test runs were completed in total. All measurements were binned according to their PEP, Table II shows the number of measurements in each bin. We

DOI: 10.1002/ett

Table II. The number of generations counted for different values of PEP.

	PEP [%]				
	0–10	10–20	20–30	30–40	40–50
$g=16$	40707	10789	509	124	56
$g=32$	18820	3407	794	662	676
$g=64$	10907	3373	695	343	150
$g=128$	9393	1372	287	183	158
$g=256$	4263	890	215	142	138

Table III. Average number packets per generation for $g=16$.

	PEP [%]				
$g=16$	0–10	10–20	20–30	30–40	40–50
n_{sent}	16.63	19.09	22.09	26.92	30.87
n_{recieved}	16.20	16.55	16.91	17.36	17.23
n_{uncoded}	15.52	13.78	12.52	10.64	9.18
n_{coded}	0.48	2.22	3.48	5.36	6.82
$n_{\text{dependent}}$	0.20	0.55	0.91	1.36	1.23

note that the uncertainty in the measurements are higher for high PEP values as fewer results were observed in those bins.

In the following Tables III–VII we have grouped the results according to the different generation sizes. For each generation n_{sent} denotes the average number of packets sent from the source before completing the generation. n_{sent} was calculated using the first and last sequence number in the generation. n_{received} denotes the average number of packets received to complete the generation i.e. including n_{uncoded}, n_{coded} and $n_{\text{dependent}}$ which denote, respectively, the coded packets, uncoded packets and linear dependent packets received.

Several trends in the tables are similar for all generation sizes. As the measured PEP increases the ratio between uncoded packets and coded packets change. This is to be expected as the number of uncoded packets sent in the initial phase is fixed, and as the PEP increases, more and more erasures must be fixed in the second phase of the systematic code. Additionally the amount of linear dependent received packets increases. This makes sense as each coded packet has a non-zero probability of being linear dependent. Inspecting n_{received} we see that the generations are typically completed using between zero and two additional packets depending on the PEP. This is in agreement with the analytical results of the coding performance presented in Reference [10].

In Figure 5, the development in throughput versus PEP is shown. The throughput approximately drops affine with the PEP, and it can be seen how the higher computational complexity of the larger generations sizes affect the performance as the PEP increases and more packets must be coded. The average maximal throughput 0.395 MB/s measured lies approximately 33% below the maximal WLAN throughput without coding presented in Reference [11] on

Table IV. Average number packets per generation for $g=32$.

	PEP [%]				
$g=32$	0–10	10–20	20–30	30–40	40–50
n_{sent}	32.88	38.04	43.93	51.86	61.33
n_{recieved}	32.27	32.76	33.25	33.56	33.58
n_{uncoded}	31.32	27.88	24.61	21.30	18.28
n_{coded}	0.68	4.12	7.39	10.70	13.72
$n_{\text{dependent}}$	0.27	0.76	1.25	1.56	1.58

Table V. Average number packets per generation for $g=64$.

	PEP [%]				
$g=64$	0–10	10–20	20–30	30–40	40–50
n_{sent}	66.17	75.82	86.53	99.94	118.59
n_{received}	64.56	65.28	65.36	65.37	65.54
n_{uncoded}	62.28	54.52	49.29	44.12	35.91
n_{coded}	1.72	9.48	14.71	19.88	28.09
$n_{\text{dependent}}$	0.56	1.28	1.36	1.37	1.54

Table VI. Average number packets per generation for $g=128$.

	PEP [%]				
$g=128$	0–10	10–20	20–30	30–40	40–50
n_{sent}	130.66	150.42	171.63	199.21	237.15
n_{recieved}	128.67	129.39	129.56	129.68	129.69
n_{uncoded}	125.99	109.99	95.12	81.68	72.52
n_{coded}	2.01	18.01	32.89	46.32	55.48
$n_{\text{dependent}}$	0.67	1.39	1.56	1.68	1.69

Table VII. Average number packets per generation for $g=256$.

	PEP [%]				
$g=256$	0–10	10–20	20–30	30–40	40–50
n_{sent}	260.69	301.28	340.35	398.73	465.49
n_{received}	256.79	257.59	257.50	257.47	257.61
n_{uncoded}	252.34	219.57	186.49	153.87	120.94
n_{coded}	3.66	36.43	69.51	102.13	135.06
$n_{\text{dependent}}$	0.79	1.59	1.50	1.47	1.61

DOI: 10.1002/ett

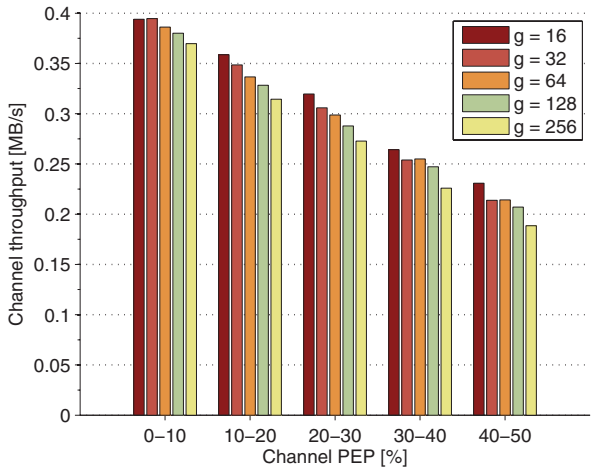


Figure 5. Throughput for different strategies with a single sink as a function of the PEP.

the same type of device. This stresses that the coding operations are not 'free' and further work should be done to optimise their implementation.

Using the measured ratios given in Tables III–VII we are able to calculate the energy consumption of the different schemes. To compute this we use the energy consumed due to the sending and receiving and the energy consumed due the coding operations. We use the values given in Reference [7] for energy receiving and sending data over WLAN and the energy measurement given in Section 5.1. Figure 6 shows the development in energy per byte spent as the PEP increases. The lower generation sizes perform worse, in terms of energy consumption, compared to the larger generation sizes, especially for higher PEPs. Thus in this case

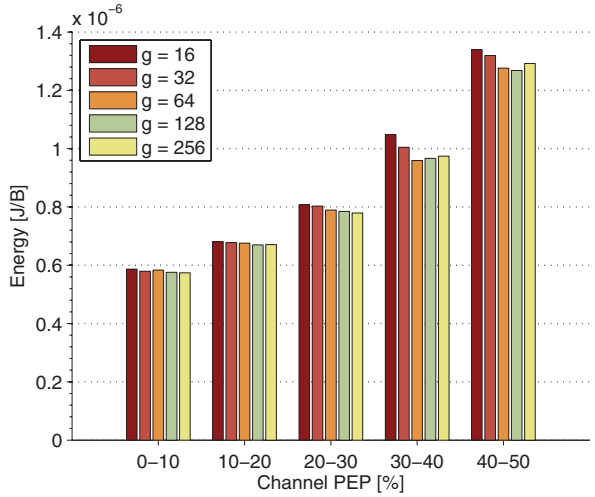


Figure 6. Energy for different strategies with a single sink as a function of the PEP.

we see that the higher overhead in terms of linear dependent packets for small generation sizes outweighs the higher energy consumption of the encoding/decoding of the larger generation sizes.

Based on these results we see an interesting trade-off between energy and speed. Where small generation sizes deliver high throughput, but higher generation sizes deliver a better energy per byte ratio. We also stress that the prototype did not use any form of feedback from the sink to the source. If feedback was introduced e.g. on a per generation basis the lower generation sizes may loose its advantage in speed as it would need a higher amount of signalling.

5.3. *Coding and cooperation in a network*

To observe the effect of recoding we need to test a setup where the sinks are cooperating by forwarding recoded packets to each other. The simplest approach is to let each sink recode and forward whenever it receives a packet from the source, with some fixed probability p_R. This probability should be chosen in accordance with the PEP of the sinks in the cluster, and will also depend on what parameters we wish to optimise, here we have chosen 5% and 10%. This simple protocol allows us to observe the effect of recoding. Enabling recoding should offload the source, in terms of both energy and computations, by moving some of the coding to the cluster. The effect should be biggest when the PEP is high, and especially visible in cases where the channel between the source and a sink is weak, but the channels between cooperating sinks are strong. In this case a fixed p_R was used; however, for a real protocol implementation it will be important to consider when a sink has enough information to be a useful 'recoder'.

In the test we measured from the first packet was received until the packet which completes the decoding of the generation was received. This allows us to obtain the performance as if we had a perfect feedback-channel and feedback scheme. The sink records the following parameters; time per generation, PEP, total packets, uncoded packets, coded packets, linear dependent packets, relayed coded packets, relayed linear dependent packets.

As in the previous setup the test was conducted using three Nokia N95s, one source and two sinks. Packets were coded using the generation size $q = 64$ and a packet size of 1200 bytes. Approximately 9.000 test runs were completed in total for each p_R. All measurements were binned according to their PEP, Table VIII shows the number of measurements in each bin. We note that the uncertainty in the measurements are higher for high PEP values as fewer results were observed in those bins.

DOI: 10.1002/ett

Table VIII. The number of generations counted for different values of PEP.

	PEP [%]				
	0–10	10–20	20–30	30–40	40–50
$g = 64$ $p_R = 5\%$	8091	891	176	53	63
$g = 64$ $p_R = 10\%$	8114	1355	266	140	109

As in the previous tables rows starting with n denotes data originating from the server, in addition the rows starting with c i.e. c_{coded} and $c_{\text{dependent}}$ denotes, respectively, the useful coded packets and linear dependent packets received from the relay. As shown in the Tables IX and X we can observe the same tendencies as in the non-cooperation case for the source to sink communication. However, in the relay communication we can observe, as expected, that the relay becomes an increasingly better source of information as the PEP increases. The main reason for this, is that the relay will only be able to repair the uncorrelated losses, that is losses which occurred only at the other sink. In addition since RLNC is used the relay will randomly pick which packets to recode, further minimising the probability of selecting an useful packet. However, as the PEP increases so does the probability that one relay has useful packets to offer the second relay. This tendency can be seen for $p_R = 5\%$ where the ratio of useful packets changes from 19% to 73% and for the $p_R = 10\%$ case where the ratio changes from 21% to 58%.

These results indicate that a successful relay protocol should be able to adapt to the current channel conditions in order to avoid sending unnecessary redundant data e.g. when the PEP is low. We do however also see that any packets coming from the relay will aid the source, in the way that the source needs to transmit a relative lower overhead to overcome the channel PEP as part of the redundancy is now coming from the relays.

Table IX. Average number packets per generation for $g = 64$. Using 5% p_R.

	PEP [%]				
$g = 64$	0–10	10–20	20–30	30–40	40–50
n_{sent}	67.12	73.44	80.30	87.75	93.46
n_{received}	63.45	64.65	64.21	62.69	62.72
n_{uncoded}	61.40	57.01	52.57	48.15	44.49
n_{coded}	2.04	5.92	9.98	13.64	16.92
$n_{\text{dependent}}$	0.01	1.72	1.66	0.90	1.31
c_{coded}	0.56	1.07	1.45	2.21	2.59
$c_{\text{dependent}}$	2.46	1.87	1.32	0.92	0.92

Table X. Average number packets per generation for $g = 64$. Using 10% p_R.

	PEP [%]				
$g = 64$	0–10	10–20	20–30	30–40	40–50
n_{sent}	67.48	73.37	80.36	87.31	93.11
n_{received}	62.77	63.75	62.99	63.29	62.66
n_{uncoded}	60.35	56.06	52.14	50.09	47.46
n_{coded}	2.41	6.17	9.19	11.24	13.47
$n_{\text{dependent}}$	0.01	1.52	1.66	1.96	1.73
c_{coded}	1.24	1.77	2.67	2.67	3.07
$c_{\text{dependent}}$	4.65	3.98	3.18	2.89	2.18

6. DISCUSSION

When NC is used several parameters must be defined, these parameters influences the performance in terms of coding throughput, network throughput, decoding delay, etc. A good choice will depend on the type of application, the target platform, the network characteristics, etc. In the following we will discuss these parameters and how they can be selected.

6.1. Parameter considerations

The field size, q, defines the size of the field over which coding operations are performed and also the size of the data symbols. From a network perspective a high q is preferable as it gives a low probability that packets are linear dependent. However, a high q can result in low coding throughputs which can be problematic in many applications and can influence the energy consumption negatively. Here we have considered only $q = 2$, as this is the only choice that have currently been shown to be practical realisable on the target platform [12, 7, 10, 13]. On other platforms this choice can be less restricted [14, 15].

The generation size, g, defines the number of packet in each generation and thus the number of packets coded together. A low g gives a high coding throughput but a higher probability of linear dependent packets, a higher g gives a lower probability of linear dependent packets but a lower coding throughput [10]. Thus the choice of g is a trade-off between network performance and coding throughput. Additionally a higher g increases the decoding delay, which is important for some applications, e.g. audio and video streaming.

DOI: 10.1002/ett

The packet size, m, defines the number of symbols per packet. A higher m increases the coding throughput [14, 7]. However, a high m can be impractical as it can result in fragmentation at lower layers. If one coded packet is fragmented into several frames, and one of these frames is lost, the rest of the frames will be useless.

A good choice of these parameters depend on the application data. For bulk data transfer the requirements to decoding delay is loose, the file(s) will not be usable until everything is decoded. However, if relatively large amounts of data is to be transferred quickly, it is important that the coding throughput is high, in order to reduce the usage of computational resources. For audio and video streaming a very important requirement is a low decoding delay, but the requirement can be loosened by increasing the playback buffer size. This is not possible for VOIP and video conferencing as it would introduce lag in the communication.

6.2. Protocol considerations

The challenge of ensuring reliable multicast transmission in arbitrary networks is an open problem with no solution within sight. To create a usable application this problem needs to be addressed at least for the scenario where the application is deployed.

The solution in the prototype is simply to overshoot, thus sending additional packets for each generation in order to compensate for packet losses. Such an approach is for example used in Multimedia Broadcast and Multicast Services (MBMS) system where the overshooting is tuned based on infrequent feedback from nodes in the network, such that a predefined fraction of the sinks can decode. Because the overshooting is fixed at some level the sinks that experience a packet loss below this level will be able to decode the data, while the remaining sinks will not. This approach is simple and works well if the sinks have relatively uniform and static channel conditions. If the feedback channel is weak or non-existing this may be the only available solution.

Another approach is to let the sinks request more data if they need it. The source sends data from a generation, alternatively it also send some overhead, and then proceeds to the next generation. If any of the sinks were not able to decode the generation they signal that they needs additional information which the source sends. This approach adapts better to changing channel conditions and as such can utilise the channel better, however, the feedback from the sinks introduces the exposure problem [16] and the crying baby problem [17]. Thus this approach works best if the sinks have relatively uniform channel conditions, and if the number of sinks is moderate.

As the links to sinks are independent they will hold different information when the source has transmitted data. Thus an interesting approach is to let the sinks cooperate and thus exploit the connection diversity. Instead of a sink requesting additionally data specifically from the source, any node that received the request could respond, thus more than one node could potentially attempt to answer the request, which would introduce the implosion problem [16]. One of the main drawbacks of this approach is the high complexity it introduces, one technique to remedy this could be the NC. Additionally if done correctly it could potentially allow for transmission in partially connected networks.

Thus in addition to the overall system operation there is several problems reliable transmission in a broadcast network that must be addressed, namely the implosion, exposure, and crying baby problem. Furthermore, a range of protocol functionality is necessary or beneficial, such as service discovery, cluster forming, multi-hop routing, connection loss and reconnection, TCP friendliness, and security, especially when partial connected mesh networks and cooperation is considered.

7. CONCLUSION

In this paper we have introduced a demo application for mobile phones, PictureViewer, that via network coding enables a user to share content with several other users. The application itself is simple but it demonstrates that network coding does not necessarily result in high complexity or overwhelming energy consumption. The implemented algorithms are designed to allow for high coding throughput, therefore a binary Galois Field and a systematic random code was used.

The achieved encoding, recoding and decoding throughput are relatively high when compared with the throughput of the WLAN. As the generation increases the computational complexity increases, as a result the coding throughput decreases and the energy consumption increases. Not surprisingly the systematic approach is considerably faster, especially when the PEP is low.

When the source is encoding and the sinks are decoding the rate at which the source transmits is significantly reduced when the generation size is increased. The energy consumption depends mostly on the PEP, but is also influenced by the generation size. In the test setup a generation size of 64 appears to achieve a good trade-off between coding throughput and linear dependence, if we observe the energy consumption. On platforms with higher computational

DOI: 10.1002/ett

capabilities and/or lower network throughput it is likely that a higher generation size would be a good choice.

The use of recoding was beneficial when the observed PEP was increased. For low values of PEP a large ratio of sent packets where linear dependent. However, as the PEP increased this ratio changed. This indicates that protocols using recoding in this type of networks, should be aware of the channel conditions using recoding only when detecting a certain level of PEP. The use of recoding at the relays was however in all cases beneficial for offloading the source when compared to the non-recoding case.

ACKNOWLEDGEMENTS

This work was financed by the CONE-FTP project grant No. 09-066549, by the Danish Ministry of Science, Technology and Innovation. The authors would like to thank Nokia for providing technical support as well as mobile phones. Special thanks to Mika Kuulusa, Gerard Bosch, Harri Pennanen and Nina Tammelin from Nokia.

REFERENCES

1. Ahlswede R, Cai N, Li SYR, Yeung RW. Network information flow. *IEEE Transactions on Information Theory* 2000; **46**(4):1204–1216.
2. Ho T, Koetter R, Medard M, Karger D, ros M. The benefits of coding over routing in a randomized setting. *Proceedings of the IEEE International Symposium on Information Theory, ISIT '03*, 2003. URLciteseer.ist.psu.edu/ho03benefits.html.
3. Médard M, Koetter R. Beyond routing: an algebraic approach to network coding. *INFOCOM*, 2002.
4. Katti S, Rahul H, Hu W, Katabi D, Medard M, Crowcroft J. Xors in the air: practical wireless network coding. *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '06)*, ACM Press, 2006; 243–254.
5. Park JS, Gerla M, Lun DS, Yi Y, Medard M. Codecast: a network-coding-based ad hoc multicast protocol. *Wireless Communications, IEEE [see also IEEE Personal Communications]* 2006; **13**(5):76–81, doi:10.1109/WC-M.2006.250362.
6. Jacobsen R, Jakobsen K, Ingtoft P, Madsen T, Fitzek F. Practical evaluation of partial network coding in wireless sensor networks. *4th International Mobile Multimedia Communications Conference (MobiMedia 2008)*, ICTS/ACM: Oulu, Finland, 2008.
7. Heide J, Pedersen MV, Fitzek FHP, Larsen T. Cautious view on network coding - from theory to practice. *Journal of Communications and Networks (JCN)* 2008; **10**(4):403-411.
8. Fragouli C, Boudec J, Widmer J. Network coding: an instant primer. *SIGCOMM Computer Communication Review* 2006; **36**(1):63–68.
9. Xiao M, Aulin T, Médard M. Systematic binary deterministic rateless codes. *Proceedings IEEE International Symposium on Information Theory*, 2008.
10. Heide J, Pedersen MV, Fitzek FH, Larsen T. Network coding for mobile devices - systematic binary random rateless codes. *The IEEE International Conference on Communications (ICC)*, Dresden, Germany, 2009.
11. Pedersen M, Perrucci G, Fitzek F. Energy and link measurements for mobile phones using IEEE802.11b/g. *The 4th International Workshop on Wireless Network Measurements (WiNMEE 2008) - in Conjunction with WiOpt 2008*, Berlin, Germany, 2008.
12. Pedersen MV, Fitzek FH, Larsen T. Implementation and performance evaluation of network coding for cooperative mobile devices. *IEEE Cognitive and Cooperative Wireless Networks Workshop*, IEEE, 2008.
13. Shojania H, Li B. Random network coding on the iphone: fact or fiction? *NOSSDAV '09: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM: New York, NY, USA, 2009; 37–42, doi:http://doi.acm.org/10.1145/1542245.1542255.
14. Shojania H, Li B. Parallelized progressive network coding with hardware acceleration. *Quality of Service, 2007 Fifteenth IEEE International Workshop on*, 2007; 47–55, doi:10.1109/IWQOS.2007.376547.
15. Vingelmann P, Zanaty P, Fitzek FH, Charaf H. Implementation of random linear network coding on opengl-enabled graphics cards. *European Wireless 2009*, Aalborg, Denmark, 2009.
16. Radoslavov P, Papadopoulos C, Govindan R, Estrin D. A comparison of application-level and router-assisted hierarchical schemes for reliable multicast. *Networking, IEEE/ACM Transactions on* 2004; **12**(3):469–482, doi:10.1109/TNET.2004.828950.
17. Holbrook HW, Singhal SK, Cheriton DR. Log-based receiver-reliable multicast for distributed interactive simulation. *SIGCOMM Computer Communication Review* 1995; **25**(4):328–341, doi:http://doi.acm.org/10.1145/217391.217468.