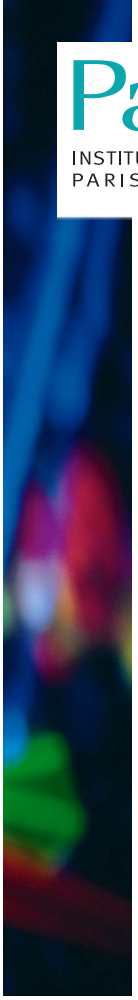
# Reliability and deployment issues of network coding in wireless networks

Paul-Louis Ageneau



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2017-ENST-0007

EDITE - ED 130

# Doctorat ParisTech

## T H È S E

**pour obtenir le grade de docteur délivré par**

## TÉLÉCOM ParisTech

### Spécialité « Informatique et Réseaux »

*présentée et soutenue publiquement par*

**Paul-Louis AGENEAU**

le 28 février 2017

# Fiabilité et problèmes de déploiement du codage réseau dans les réseaux sans fil

Directeur de thèse : **Mme Nadia BOUKHATEM**

**Jury**

| | |
|---|---|
| **M. André-Luc BEYLOT**, Professeur, IRIT, ENSEEIHT | Rapporteur |
| **M. Yacine GHAMRI-DOUDANE**, Professeur, L3i, Université de La Rochelle | Rapporteur |
| **M. Mario GERLA**, Professeur, University of California, Los Angeles | Examinateur |
| **Mme Thi Mai Trang NGUYEN**, Maître de Conférences HDR, LIP6, UPMC | Examinateur |
| **Mme Megumi KANEKO**, Maître de Conférences HDR, NII, Tokyo | Examinateur |
| **M. Michel BOURDELLÈS**, Ingénieur, Thales Communications | Examinateur |
| **M. Philippe MARTINS**, Professeur, Télécom ParisTech | Examinateur |

**TÉLÉCOM ParisTech**
École de l'Institut Mines-Télécom - Membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - http://telecom-paristech.fr

T H È S E

# Acknowledgements

I would like to express my most sincere thanks to my supervisor Nadia Boukhatem for her excellent guidance during my PhD years. I am genuinely thankful for the flawless support she has given me despite hardships and obstacles.

I would also like to thank Prof. Mario Gerla for inviting me to visit him at UCLA and for giving me two great opportunities to work with his team.

Moreover, I would like to express my gratitude and appreciation to Prof. André-Luc Beylot from IRIT-ENSEEIHT and Prof. Yacine Ghamri-Doudane from University of La Rochelle for taking the time to review the manuscript.

I would like to extend my gratitude to Prof. Philippe Martins, Dr. Thi Mai Trang Nguyen, Dr. Megumi Kaneko, and Dr. Michel Bourdellès for the interest they have been showing in my work and for accepting to be members of my examination commitee.

Finally, I thank my family, friends, and colleagues for their support, their help, and their encouragements. Without them being so supportive, I could never have achieved this work.

# Abstract

Even if packet networks have significantly evolved in the last decades, packets are still transmitted from one hop to the next as unalterable pieces of data. Yet this fundamental paradigm has recently been challenged by new techniques like network coding, which promises network performance and reliability enhancements provided nodes can mix packets together.

Wireless networks are nowadays ubiquitous and rely on various network technologies such as WiFi and LTE. They can however be unreliable due to obstacles, interferences, and these issues are worsened in wireless mesh network topologies with potential network relays. In this work, we focus on the application of intra-flow network coding to unicast flows in wireless networks. The main objective is to enhance reliability of data transfers over wireless links, and discuss deployment opportunities and performance.

First, we propose a redundancy lower bound and a distributed opportunistic algorithm, to adapt coding to network conditions and allow reliable data delivery in a wireless mesh. We believe that application requirements have also to be taken into account. Since network coding operations introduce a non negligible cost in terms of processing and memory resources, we extend the algorithm to consider the physical constraints of each node while still guaranteeing data delivery at destination.

Then, we study the interactions of intra-flow coding with TCP and its multi-path extension MPTCP. Network coding can indeed enhance the performances of TCP, which tends to perform poorly over lossy wireless links. We investigate the pratical impact of fairness issues created when running coded TCP flows besides legacy non-coded TCP flows. Finally, we explore two different ways to enhance the performance of MPCTP in wireless environments: running it over network coding, and implementing the coding process directly in MPTCP while keeping it fully TCP-compatible.

# Contents


**1 Introduction** **11**

1.1 Context and objectives . . . . . 11

1.2 Contributions . . . . . 14

**2 Background on network coding** **17**

2.1 Introduction to network coding . . . . . 17

2.1.1 Definition and benefits . . . . . 17

2.1.2 Linear Network coding . . . . . 19

2.2 Network coding approaches . . . . . 19

2.2.1 Inter-flow coding . . . . . 19

2.2.2 Intra-flow coding . . . . . 20

2.3 Intra-flow coding applications . . . . . 20

2.3.1 Intra-flow coding and multicast traffic optimization . . . . . 20

2.3.2 Intra-flow coding and unicast flow reliability . . . . . 20

2.4 Intra-flow coding implementation methods . . . . . 21

2.4.1 Network coding redundancy . . . . . 21

2.4.2 Generation-based coding . . . . . 22

2.4.3 Sliding-window coding . . . . . 24

2.5 Network coding for opportunistic routing . . . . . 24

2.5.1 Definition of opportunistic routing . . . . . 25

2.6 Conclusion . . . . . 27

**I Redundancy adaptation** **29**

**3 Dynamic redundancy** **31**

3.1 Objective and Motivations . . . . . 31

3.2 Redundancy control mechanisms . . . 32
3.2.1 Static redundancy . . . 32
3.2.2 Loss rate measurement . . . 33
3.2.3 Explicit feedback . . . 34
3.2.4 Conclusion . . . 36
3.3 Redundancy estimation . . . 36
3.3.1 Generation loss probability bound . . . 37
3.3.2 Redundancy lower bound . . . 38
3.3.3 Non-innovative combinations leading to intrinsic loss . . . 39
3.4 Distributed adaptive redundancy control . . . 40
3.4.1 Network model . . . 40
3.4.2 Algorithm at relays . . . 41
3.4.3 Redundancy at the source . . . 42
3.4.4 On-the-fly recoding vs delayed recoding . . . 42
3.5 Simulation results . . . 43
3.5.1 Loss bound evaluation . . . 43
3.5.2 Distributed adaptive redundancy evaluation . . . 44
3.5.3 Resilience to coding/no coding nodes . . . 44
3.5.4 Impact of delayed recoding . . . 46
3.6 Conclusion . . . 48

**4 Taking constraints into account** **49**
4.1 Introduction . . . 49
4.2 Taking constraints into account . . . 49
4.3 State of the art . . . 50
4.4 Constraint-aware Network Coding Model . . . 50
4.4.1 General principle . . . 51
4.4.2 Redundancy Selection Algorithm . . . 51
4.4.3 Algorithm at relays . . . 52
4.4.4 Special case of the source . . . 54
4.4.5 Special case of neighbors of the destination . . . 54
4.5 Simulation results . . . 54
4.5.1 Effect of the contribution factor . . . 54
4.5.2 Impact of nodes with lower contribution factors . . . 55

4.6 Conclusion . . . 56

**II Network coding and TCP** **59**

**5 Fairness and interaction between network coding and TCP** **61**

5.1 Objectives . . . 61
5.2 TCP and fairness considerations . . . 61
5.3 Sensitivity to losses and unfairness . . . 62
5.3.1 Link and congestion losses . . . 63
5.3.2 Case of competing flows . . . 63
5.4 Measuring fairness . . . 66
5.4.1 Jain's fairness index . . . 67
5.4.2 Fair allocation with coded and non-coded flows . . . 67
5.4.3 Formalization . . . 67
5.4.4 Modified fairness index . . . 69
5.5 Conclusion . . . 70

**6 Increasing reliability of MPTCP over network coding** **73**

6.1 Objectives . . . 73
6.2 Multi-path TCP over network coding . . . 74
6.3 Emulated network setup . . . 75
6.4 Performance evaluation . . . 76
6.4.1 Influence of loss rate and redundancy . . . 76
6.4.2 Influence of generation size . . . 78
6.4.3 Influence of Round-Trip Time . . . 79
6.5 Conclusion . . . 79

**7 Integration of network coding in the MPTCP protocol** **81**

7.1 Objectives . . . 81
7.2 Network coding integration in MPTCP . . . 82
7.3 Principle . . . 84
7.3.1 Activation . . . 85
7.3.2 Coding . . . 86
7.3.3 Decoding . . . 87

7.3.4 Feedback mechanism . . . 87
7.4 Implementation details . . . 87
7.4.1 Data Sequence Signal (DSS) option . . . 88
7.4.2 Component size . . . 90
7.5 Performance evaluation . . . 90
7.5.1 Scenario . . . 90
7.5.2 Results . . . 91
7.6 Conclusion . . . 91

**8 Conclusion** **93**

**A Publications** **97**
A.1 Collaborations . . . 97
A.2 Articles . . . 97
A.3 Deliverables . . . 98

**B Redundancy bound derivation** **99**
B.1 Chernoff bound . . . 99
B.2 Bound on $r$ . . . 100

# Chapter 1

# Introduction

## 1.1 Context and objectives

Since the early 1960s computer networks and their usage have drastically changed, whereas packet transmission principles have surprisingly stayed roughly the same.

Even if we nowadays use wireless access most of the time, packets are still transmitted from one hop to the next as unalterable pieces of data in most situations. Yet this fundamental paradigm has recently been challenged by new techniques like network coding, which promises network performance enhancement given the assumption that nodes can mix packets together.

Wireless networks are widely used today with WiFi 802.11 and 3G/4G/LTE as predominant technologies connecting consumer mobile devices. In the future, mobile devices will need connectivity from everywhere, smartphones will generate even more wireless traffic, everyday objects will be connected to wireless networks, cars and drones will need reliable transmissions with ground stations and between them.

Yet, wireless networks still suffer from high loss rates due to obstacles, interferences, or movements. The link quality can drop and cause packet loss on the channel. MAC layer protocols implement retransmissions in order to partially solve this issue, however, losses can still happen at upper layers after a number of transmission retries.

These issues are worsened in wireless mesh network topologies with potential network relays. The increased number of relays not only increases interferences, but the multiple wireless hops lead to increased loss at destination. With these topologies, MAC layer protocols can be inefficient at preventing a sufficient proportion of losses to make the network reliable enough for most applications.

Network coding deployment for wireless networks has been subject to a growing attention from researchers since its introduction in [1]. It has been acknowledged as a way to improve capacity and reliability in computer networks.

In particular, wireless networks, thanks to their inherent broadcast nature and overhearing capability, are considered as very suitable environments for applying network coding.

Network coding is a family of techniques based on the following principle: instead of forwarding packets one by one, network nodes combine several original packets into one *coded* packet, also called *combination*, before transmitting it. The process of combining the packets is

referred to as *coding*, and the process of retrieving original packets from coded packets is called *decoding*.

Depending on its applications, the benefits are multiple: thanks to network coding, the number of transmissions required to transmit data across a network can be substantially reduced, packet loss can be prevented and flow reliability can be increased.

Network coding techniques can be divided in two complementary families:

- Inter-flow coding: the goal is to reach maximal network capacity in topologies where conventional forwarding fails to do so, by coding together packets from different flows going through a node.
- Intra-flow coding:
  - For multicast flows: like inter-flow coding, the goal is to reach the maximal capacity in the multicast tree compared to traditional multicast.
  - For unicast flows: the goal is to enhance reliability of lossy networks by combining packets from the same flow, network coding acts as a hop-by-hop erasure scheme.

In this work, we primarily focus on the application of intra-flow network coding to unicast flows in wireless networks. The main objective is to consider network coding applications to enhance reliability of data transfers over wireless links, and to discuss deployment opportunities and performance.

One of the main features of intra-flow network coding is to prevent losses. With intra-flow coding, the original packets can be decoded when enough combinations are received, whichever combinations are actually received. It acts as a hop-by-hop erasure code, enabling to recover packets in the presence of packet losses.

This property can also increase reliability in topologies where multiple potential paths are available, for instance wireless meshes, since the end-to-end flow can be switched seamlessly to the active paths when one of the paths fails.

When intra-flow network coding is deployed for unicast flows, it is necessary to guess the number of combinations to generate at the source and at relays, in order for the destination to be able to decode the original packets. The ratio between sent combinations and original packets is called the redundancy factor, and it is therefore a crucial parameter for intra-flow network coding.

If redundancy is too low, it is not sufficient to recover the losses. The flow might actually perform worst than a flow running without coding, due to entire groups of packets being discarded because they are not decodable, causing significant grouped losses. If redundancy is too high, network overhead is increased and network tends to be congested for no benefit in terms of application performance.

For intra-flow network coding to perform well, the redundancy factor should be tuned according to network characteristics. In addition, adapting redundancy should be considered if network conditions are subject to changes.

However, even when information on link quality is available, tuning the redundancy consequently is not trivial, and several works rely on empirical formulas leading to over-redundancy situations. In this work, we propose a redundancy model to easily evaluate a possible redundancy bound.

We believe that choosing the redundancy value should take into account not only the network characteristics, but also the application requirements and its tolerance in terms of loss rate. Thus, we derive a recommended minimal redundancy value for a link taking into account the maximum loss rate the application may tolerate.

We then extend the model to allow redundancy adaptation for a whole network. We propose a distributed algorithm for each node in a wireless network to set the redundancy factor and provide a reliable data delivery for the unicast flows. A distributed approach has the advantage to be feasible in practice compared to a centralized approach.

Coding and decoding operations induce a noticeable processing overhead. They require a noticeable amount of processing power and memory space compared to traditional packet forwarding. For mobile nodes, it also means more battery consumption.

Therefore, the capacity of a node to contribute to the network coding process depends on its hardware capabilities and on its current status. For instance, it is more optimal to have the coding operations supported by a node connected to the mains power supply than by another node operating on low battery.

The idea is, in a collaborative mesh network, to push traffic to the nodes which are the most able to code. In our work, we propose to take nodes constraints into account in the distributed redundancy estimation scheme.

As stated above, network coding is a promising approach for improving throughput, reliability and robustness for wireless networks. However and despite its potential, we still seem far from seeing widespread network coding implementations and deployments across networks.

Some authors argue that a major reason is that it is not clear how to naturally add network coding to current network systems, how to incrementally deploy it, and how network coding will behave in the wild.

The interaction of network coding with TCP is one of the current deployment issues, because TCP is nowadays the predominant transport protocol in the Internet.

It is well documented that TCP performs poorly over lossy links, since it interprets packet losses as congestion signals [2]. Link loss are extremely uncommon on wired channels, so traditionally, the only possible source of loss is network congestion.

Since intra-flow network coding is able to enhance flow reliability and performance, it seems perfectly suited to run TCP on top to boost performance. Indeed, network coding allows to mask link losses, preventing the need for retransmissions that TCP could interpret as congestion signals and preventing useless congestion window decrease.

However, network coding deployment with TCP also raises concern related to flow fairness. Normally, TCP flows share the available bandwidth thanks to their congestion control algorithm. Yet, most of the time, the congestion signal is packet loss. Intra-flow coding hides losses from TCP independently of their cause, so congestion losses are hidden just like link losses. Therefore, TCP flows running over network coding could take an unfairly big part of the available bandwidth when running beside legacy non-coded flows. In the worst cases, non-coded TCP flows could even starve and fail.

We address, in this work, the fairness issue between coded TCP flows and non-coded TCP flows and define a new fairness index to study the implications of running parallel coded and non coded TCP flows.

Multi-path TCP (MPTCP) is a recently standardized TCP extension. It aims at spreading a single TCP connection over several physical paths. Its interest has been demonstrated for data centers, and also for mobile devices featuring multiple radio interfaces, like a smartphone with WiFi and LTE. Indeed, it showcases several advantages: it improves connectivity and quality of service, increases throughput by allowing the use of multiple interfaces for data transfer, and seamlessly handles handover and traffic offload from congested radio access networks [3].

Since it is based on TCP and relying on the same mechanisms, MPTCP is also subject to the similar drawbacks, especially sensitivity to link loss, which can be an issue in scenarios with multiple radio interfaces. In this work, we study the benefits of running MPTCP over network coding.

Moreover, we investigate a network coding based solution to the head-of-line blocking issue for MP-TCP. The issue happens when the global MPTCP window cannot move forward since packets scheduled on the failing path are missing. In this work, we propose a protocol sending coded segments over TCP subflows. This approach allows to overcome head-of-line blocking while keeping full TCP retro-compatibility.

## 1.2 Contributions

In this work, we propose the following contributions:

First, in chapter 3, we derive a minimal redundancy bound to set the network coding redundancy according to the link quality and the targeted maximum application loss rate. We then propose a distributed algorithm for redundancy adaptation and reliable data delivery. The algorithm allows the data producer to opportunistically make use of multiple available paths to route the coded packets to destination while offering on optimized redundancy control and offering an overall reliable data delivery.

Since network coding operations can show a non-negligible cost in terms of computing, storage and power consumption, we also present in chapter 4 an extension of the algorithm which considers the capacity of each node to contribute to the encoding operations. The constraints are taken into account when forwarding the packet combinations while guaranteeing decoding at destination.

Then, we study the interaction of network coding with TCP and its extension MPTCP.

In chapter 5, we study the impact of fairness issues happening when non-coded TCP flows run besides coded TCP flows. In order to evaluate fairness properly, we introduce a new fairness index.

In the following chapters 6 and 7, we explore two different approaches to enhance MPTCP performance with intra-flow network coding: first by running MPTCP over network coding, then by implementing network coding directly in MPTCP.

Since it is based on TCP and relying on the same mechanisms, MPTCP is also subject from similar drawbacks, especially sensitivity to link loss. Therefore, in chapter 6, we first study the benefits of running MPTCP over network coding.

In order to go further in the investigation of network coding as a solution to the head-of-line blocking issue, we designed a practical MPTCP extension, MultiPath Coded TCP (MPC-TCP), exposed in chapter 7. The core idea is, rather than running MPTCP over network coding, to

implement directly network coding in MPTCP, while keeping a strict retro-compatibility. This aim is to increase resilience, improve overall performance, while still guaranteeing TCP retro-compatibility for middleboxes traversal.

# Chapter 2

# Background on network coding

This chapter presents basic concepts on network coding, especially linear network coding. Its objective is not in any way to compile a comprehensive survey of the network coding field, but rather to introduce the basic elements that are necessary to this work.

We will first define network coding and present related general concepts, in particular the two different approaches: inter-flow coding and intra-flow coding.

Then, we will introduce opportunistic routing, a routing principle for wireless networks that can benefit from the introduction of network coding.

## 2.1 Introduction to network coding

### 2.1.1 Definition and benefits

We can define network coding as follows [4]:

**Definition 1.** *Network coding is a technique which allows network nodes to combine several native packets into one coded packet for transmission instead of simply forwarding the original packets one by one, in order to maximize network capacity.*

In this definition, *combining* packets can refer to any method producing output packets from a set of input packet, while still carrying part of the information. In practice, combining involves bitwise XOR operations between packets, *i.e.*, logical exclusive OR between corresponding bits, or more generally linear combinations of packets.

Network coding can be seen as an alternative to packet forwarding in a network. It provides two main benefits: enhancing throughput and enhancing reliability.

It has been proposed with the benefit of maximizing network capacity, *i.e.*, the maximum traffic the network can convey from sources to destinations. The well-known butterfly topology (Fig. 2.1) typically illustrates a case where network coding clearly outperforms packet forwarding.

Let's consider two source nodes transmitting two different flows A and B in a butterfly-shaped network with three routes to two destination nodes. Each destination node wants to receive both flows, and each link is assumed to have a capacity of one packet per time unit and no loss.

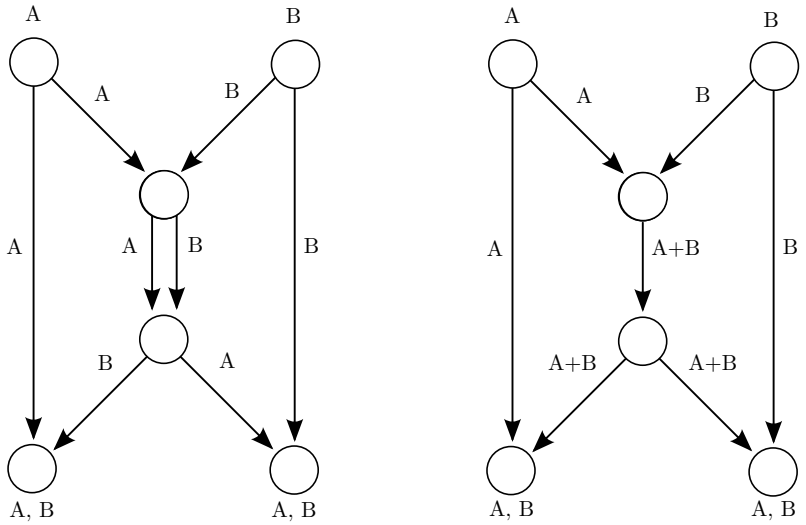



Figure 2.1: Butterfly topology without (left) and with (right) network coding

If classical forwarding of packets is used (left), only one packet can be transmitted on each link per time unit. In particular, the central link would not be able to carry both flows A and B at the same time, but only one packet of each flow per time unit. Therefore, the maximum throughput for each flow is 1.5 packets per time unit, and network capacity is 3 packets per time unit.

With network coding, the central link can perform bitwise XOR operations between packets received from A and B. In this case, it is sufficient for the two destinations to decode both A and B, simply by XORing the received packets again. Network coding allows to transmit 2 packets per time unit for each flow, enhancing network throughput to 4 packets per time unit.

In addition to this advantage for network throughput enhancement compared to traditional packet forwarding, network coding can also enhance the reliability of unicast flows.

Indeed, network coding provides benefits over retransmissions. Conventional retransmission mechanisms, *e.g.*, Automatic Repeat reQuest (ARQ), require determining whether each packet is properly received or lost, in order to perform a retransmission of this individual packet. With network coding, the source can combine packets to transmit together, then, the destination only has to get enough combinations to decode the information. Thus, the source does not care about which specific pieces of data to retransmit.

Network coding also allows to send more combinations than original packets. The destination only needs to get as many combinations as original packets to achieve decoding, therefore original data can be recovered even if some packets are lost.

This advantage can be leveraged when overhearing nodes can perform opportunistic retransmissions. With simple forwarding, neighbor nodes would have to explicitly exchange information to perform retransmission, because a missing packet at destination can only be fixed by retransmitting this exact packet. Network coding can solve this problem transparently since any innovative combination can be used to decode the original packets.

As stated above, this is however at the expense of an added decoding delay. In network coding, there is obviously a tradeoff between the cost of coding operations and the performance benefits the coding provides. The challenges when implementing applications based on network

coding is to find a balance between the cost and benefits.

### 2.1.2 Linear Network coding

Most practical works on the subject make use of linear network coding. Linear coding uses linear algebra to achieve packet coding: packets are considered as vectors and the combinations of packets are linear combinations. It has been proven that linear coding is sufficient to achieve the optimal throughput in multicast situations [5].

The coefficients chosen to compute combinations can be obtained in two manners:

- Deterministic linear coding: coefficients are deterministic and derived from a known algorithm
- Random linear coding: coefficients are randomly or pseudo-randomly generated

It has been demonstrated that random coefficients for linear network coding nearly allow to reach network capacity in broadcast transmission schemes like wireless networks using a decentralized algorithm, provided the field size is sufficiently large [6].

The larger the field size, the higher the probability for the receiver to obtain linearly independent combinations. In practice, a field of $2^8$ elements is used most of the time since it is sufficient to achieve a good decoding probability, and it allows better performance. The enhanced performance comes from the fact that symbols in this field are actual bytes. This allows to manipulate packet data directly without bit masks, and eases their manipulation as most systems operate on byte basis and not bit basis.

## 2.2 Network coding approaches

Considering the flows from which the packets are coded together, we can distinguish two main categories of approaches: inter-flow coding and intra-flow coding.

### 2.2.1 Inter-flow coding

Inter-flow network coding aims at maximizing network capacity by combining packets from different unicast flows at relays.

Packets from different flows are combined together in order to send less packets and reduce wireless transmission time. The idea is to to transmit only the necessary information accross links. The spared transmission time is available for other flows. The result is an increased throughput for each flow, and increased network capacity. The butterfly network presented in the previous section (Fig. 2.1) shows a classic example of how the available throughput can be increased.

One of the pioneering works for inter-flow network coding in wireless networks is COPE [7]. Each node taking part in COPE opportunistically combines packets belonging to different unicast flows. Like most inter-flow coding schemes, packet combinations are performed with XOR operations. COPE infers coding opportunities, *i.e.*, packets that can be coded together and sent only once, from the knowledge of packets heard by each neighbor. The information is gathered by listening transmissions from neighboring nodes.

### 2.2.2 Intra-flow coding

Intra-flow network coding enables to increase flow reliability by combining packets from a single flow together. The idea is to send combinations of outgoing packets from the same flow rather than sending the packets themselves.

At destination, Gauss-Jordan elimination is performed and all packets can be recovered if enough independent linear combinations have been received. Depending on the coding implementation, intermediary nodes can either recode packets by mixing forwarded combinations, or just forward them.

To compensate losses, more combinations than original packets can be generated. At destination, original packets can be decoded, provided enough independent combinations are received.

## 2.3 Intra-flow coding applications

Intra-flow network coding can how interesting advantages for multicast or unicast flows.

### 2.3.1 Intra-flow coding and multicast traffic optimization

When a unique source use a multicast flow to reach multiple destinations, packets follow a multicast tree. If branches of the tree have common nodes, intra-flow coding can enhance the capacity along the tree in the same way inter-flow coding can enhance network capacity, therefore increasing the possible multicast flow throughput.

To illustrate this benefit, let's consider a source node multicasting in a butterfly-shaped network with three routes to two destination nodes (Fig. 2.2). Each destination node wants to receive the entire multicasted flow, and each link is assumed to have a capacity of one packet per time unit and no loss.

If only classic forwarding of packets is allowed (left), only one packet can be transmitted on each link per time unit. Therefore, the maximum throughput for the flow is 1.5 packets per time unit to each destination, *i.e.*, 3 packets per time unit in total.

With intra-flow network coding (right), similarly to the inter-flow butterfly topology case, the total throughput can be enhanced to 4 packets per time unit.

### 2.3.2 Intra-flow coding and unicast flow reliability

In its common use case, intra-flow coding can enhance reliability of unicast flows on lossy networks by coding packets from the same flow together. Since it allows generating a virtually infinite number of combinations from a set of outgoing packets, it can be considered as a form of fountain code, a class of erasure codes.

The fountain codes like Luby Transform code (LT code) [8] and the more efficient Raptor code [9] are indeed well-known solutions to the problem of packet transmission over lossy links, as they are the first practical realizations of fountain codes that achieve near-optimal erasure correction.

However, typical usage cases of fountain codes are quite different. Fountain codes are indeed

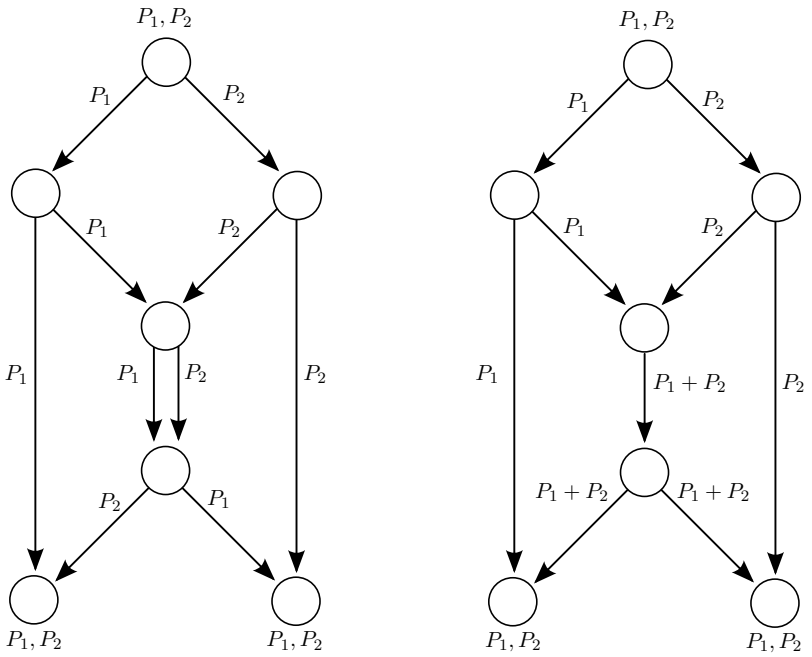



Figure 2.2: Multicast without (left) and with (right) network coding

designed to operate end-to-end only, contrary to network coding which allows to recoded on relay nodes, and they imperatively need a feedback channel to notify the source that the entire block of data has been decoded. Hop-by-hop network coding is clearly a better choice than end-to-end coding to deal with link loss, since packet loss happen on each link.

Moreover, they aim at being light in terms of processing power rather than being efficient in terms of coding. In pratice, their efficiency is random and the destination needs way more coded packets than original packets to be able to decode.

## 2.4 Intra-flow coding implementation methods

### 2.4.1 Network coding redundancy

Intra-flow network coding allows to send more combinations than original packets in order to increase flow reliability. Thus, a flow can sustain a number of packet loss while still conveying the original data. The enforced ratio between sent combinations and original packets, called the redundancy factor, is a critical parameter.

The redundancy factor should be large enough to compensate link losses and guarantee enough combinations at reception to ensure the packets are decoded. However, setting it too high can lead to useless overhead and network congestion.

For instance, if the sender has 3 packets to send, it may send 4 combinations in order to protect the packets from loss (Fig. 2.3). In that case, the redundancy factor $r$ is 4/3. If any of the combinations is lost, the destination can still decode the original packets without any difference. Note that if none of the combinations is lost, the destination simply ignores the supernumerary combination because it doesn't bring new information and is useless to the decoding process. Such a combination is called *non-innovative*.

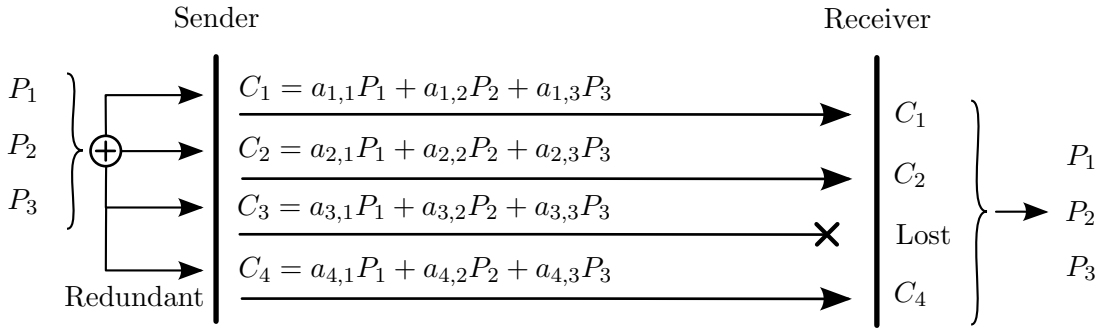



Figure 2.3: Network coding redundancy example with 3 original packets and 4 combinations (redundancy factor $r = 4/3$)

### 2.4.2 Generation-based coding

To implement a network coding system, it is necessary to identify which packets generated by the source to code together. One of the most well-known approach is called generation-based coding.

Computation needed for coding and, more particularly, decoding, primarily depends on how many packets are coded together. Due to technical constraints, in practical implementations, a common technique is to group packets in consecutive batches, called generations. Packets belonging to the same generation are coded together.

The primary goal is to reduce processing time at the source, where per packet processing is roughly in $O(n)$, and even more at destination, where it is in $O(n^2)$ (Gauss-Jordan elimination has complexity $O(n^3)$ to decode $n$ packets). The secondary goal is to reduce buffering delays, since packets of the same generation must be cached together.

Small generations allow to reduce processing time and delay. However, they also reduce the efficiency of network coding. The main reason is obvious: the destination needs enough packets from a specific generation to perform decoding of this generation. Therefore, smaller generations tend to suffer from the same drawback as packet forwarding. Traditional forwarding can be seen as a generation of size 1.

Considering they way the packet are coded together inside a generation we can distinguish two kinds of methods:

#### 2.4.2.1 Batch coding

In Batch coding, the source randomly combines packets of the same generation together (Fig. 2.4).

**Definition 2.** ***Batch coding*** *Let $m$ be the generation size and $P_1, ..., P_n$ the original packets, then for each combination $C_{g,i}$ in generation $g$, it exists $a_{g,i,1}, ..., a_{g,i,m}$ such as*

$$C_{g,i} = \sum_{j=1}^{m} a_{g,i,j} P_{mg+j}$$

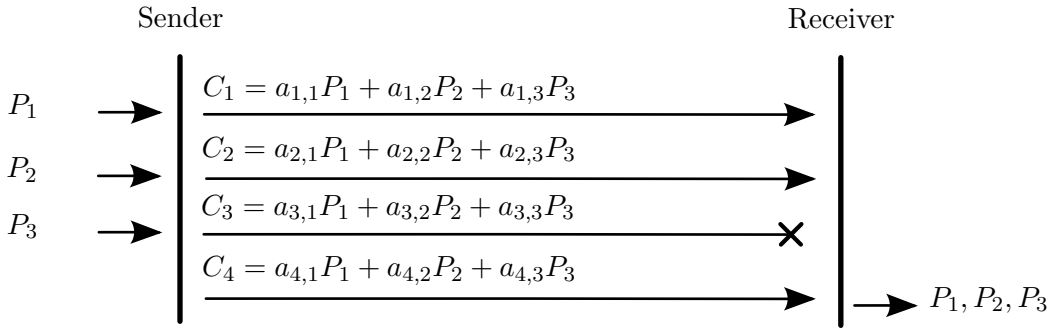



Figure 2.4: Batch coding principle with a generation size $m = 3$ and one redundant combination

#### 2.4.2.2 Pipeline coding

Pipeline coding [10] is a generation-based network coding scheme developed for low-delay transmissions. It is particularly suitable for real-time interactive and multimedia sessions. Additionnaly, it works well when combined with TCP protocol based applications. The reduced coding delay avoids TCP congestion control timeouts to be trigged and the connection to be stall.

Pipeline coding encodes and decodes packets progressively. Its principle is to store outgoing packets in a coding buffer and to send combinations from it as soon as possible. When a new packet arrives from the application, it is added to the coding buffer and one or more combinations (on average $r$, where $r$ is the redundancy factor) is immediately sent (Fig. 2.5).

When the coding buffer is full, *i.e.*, the number of packets reaches the generation size, it is cleared for a new generation. A small generation size means less computation is required and reduces decoding delays, making however the flow more sensitive to sudden increases in the loss rate.

**Definition 3.** ***Pipeline coding*** *Let $m$ be the generation size and $P_1, ..., P_n$ the original packets, then for each combination $C_{g,i}$ in generation $g$, it exists $a_{g,i,1}, ..., a_{g,i,i}$ such as*

$$C_{g,i} = \sum_{j=1}^{i} a_{g,i,j} P_{mg+j}$$

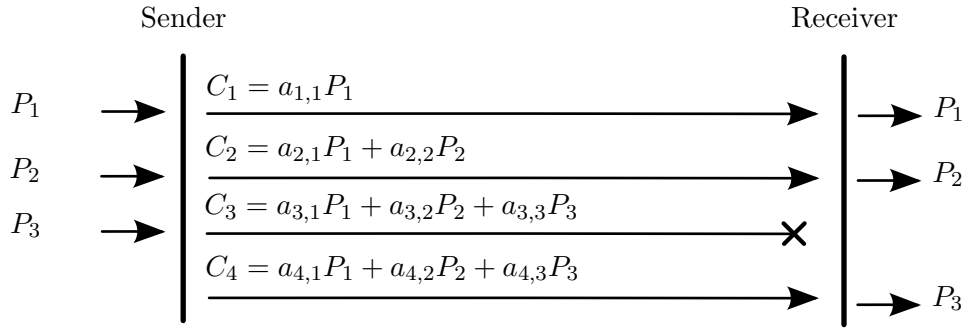



Figure 2.5: Pipeline coding principle with a generation size $m = 3$ and one redundant combination

### 2.4.3 Sliding-window coding

Sliding-window coding is an alternative to generation-based network coding. It allows to smooth the coding and decoding process and perfectly suits protocols already using a transmission window, for instance TCP (as in TCP/NC [11]), however it has the drawback of requiring a form of feedback.

The principle is to use a sliding window to code packets together. The window moves forward by removing old packets that are not to be included in the next combinations and adding new packets generated by the upper layer.

The sliding-window coding scheme needs a feedback from the decoding process to remove older packets. The packets that can be discarded from the coding process are packets that are mot needed anymore in order to complete the decoding process.

More precisely, in the Gauss-Jordan Elimination algorithm used for decoding, they are the packets that can be expressed only with more recent packets, called *seen* packets (Fig. 2.6). Formally, a packet $P_i$ is seen if and only if:

$$\exists(\alpha_{i+1}, ..., \alpha_m) \ / \ P_i = \sum_{k=i+1}^{m} \alpha_k C_k$$

Seen packets are not necessary decodable, but since they can be expressed only in the form of combinations of future packets, they will be decodable in the future even if they do not figure in incoming combinations anymore. Of course, packets that are not yet seen are called *unseen*.

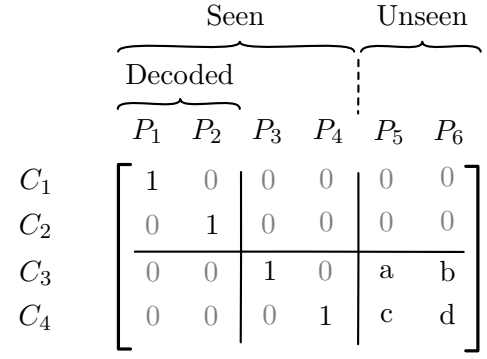



Figure 2.6: Partial decoding highlighting seen packets

## 2.5 Network coding for opportunistic routing

Applications of intra-flow network coding are numerous, and wireless networks are a particularly major field of application given the benefits network coding can offer in terms of capacity and reliability. Specifically, network coding can improve performances of opportunisting routing.

In this section, we quickly present the core principles of opportunistic routing that will be needed in our work.

### 2.5.1 Definition of opportunistic routing

Opportunistic routing [12], contrary to traditional routing, takes advantage of the broadcast nature of the wireless medium. The principle is that every neighbor can opportunistically serve as a next hop. Such a routing method is particularly suited for mesh networks.

Intra-flow network coding is particularly adapted to opportunistic routing, since it solves the issue of which packet to forward. Indeed with network coding, the only requirement for the destination is to receive enough independant combinations.

The reliability is increased because every neighbor can transmit a packet it receives or hears (Fig. 2.7). If a packet is not received successfully by a next hop, it can nevertheless be forwarded by another one.

The gain comes at virtually no cost thanks to the broadcast nature of WiFi: a packet only has to be transmitted once to have a chance to be received by any neighbor.

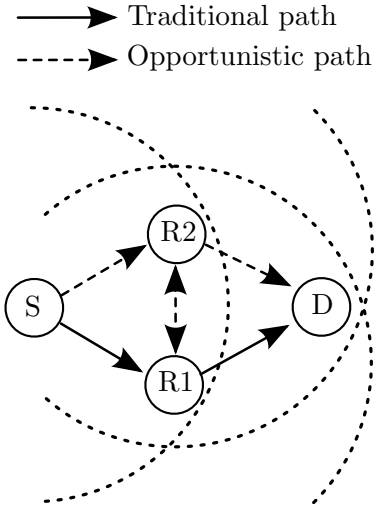



Figure 2.7: Opportunistic routing principle: neighbor node R2 can also serve as a potential relay, adding possible paths.

To highlight the reliability benefit of opportunistic routing, we take the following example: The source node S wants to transmit a packet to a destination node D, and 3 potential relays R1, R2 and R3 are available (Fig. 2.8). If we assume the loss rate to each relay is $p = 0.1$, with traditional routing, S choses only one relay (*e.g.*, R2), and the actual loss rate is 0.1. However, with opportunistic routing, the packet is lost only if it is received by none of the relays, so the actual loss rate becomes $p^3 = 0.001$, which is negligible in comparison with traditional routing.

However, opportunistic routing present a non-negligible drawback. The destination needs to receive every packet at least once, and forwarding a single packet multiple times is useless and can lead to a waste of resources. Therefore, a form of synchronization between nodes is needed to determine which packets to forward.

ExOR [13] can be considered as the first practical opportunistic routing system. To reduce the coordination overhead between candidate relays, in ExOR the packets to be transmitted are grouped into batches according to their destination node.

For each packet of the same batch, the source node selects a subset of optimal candidate forwarders, which are prioritized by closeness to the destination. Nodes in ExOR employ the

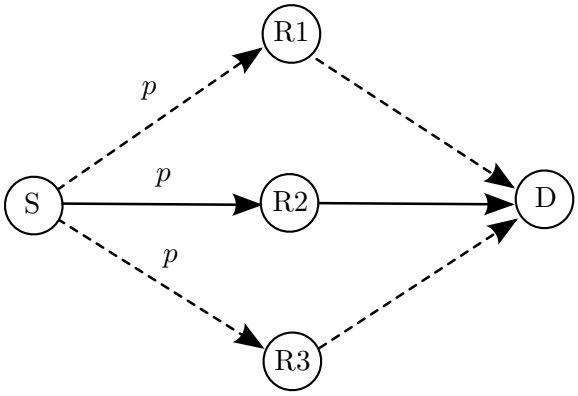



Figure 2.8: Opportunistic routing advantage: overall packet loss is reduced when multiple relays are available

ETX [14] metric to define the shortest path from the source to the destination. The list of selected forwarders, sorted by node priority, is added to the header of each packet broadcasted by the source. Hence, each node receiving a packet knows whether it has to participate in the forwarding process or not, and its position in the forwarding schedule.

The first practical system to combine intra-flow linear coding with opportunistic routing is MORE [15]. In MORE, packets are grouped into batches, linear combinations of packets from the same batch are sent to neighbor nodes. Upon opportunistically overhearing combinations, potential relays may generate new combinations.

In parallel, to guarantee decoding at destination, MORE implements a Stop-and-Wait Automatic Repeat Query (ARQ) mechanism with explicit control messages: the source continues generating new combinations from the same batch until it receives an acknowledgement from the destination.

Two enhancements of MORE, CodeOR [16] and SlideOR [17], have been proposed to enhance its performance. They aim at upgrading the potentially inefficient stop-and-Wait mechanism, which can be inefficient in terms of bandwidth utilization when network delay is high.

Instead, CodeOR allows the source to transmit multiple batches, called segments, so that more packets can be transferred. Moreover, intermediate nodes can start processing a new segment if possible. Besides the end-to-end acknowledgement inherited from MORE, CodeOR also deploys hop-by-hop acknowledgement which is generated by an intermediate node to inform its upstream nodes that it have received a full segment and forward it to the destination without any further assistance from the upstream.

SlideOR [17] an enhancement for CodeOR. In the manner of a sliding-window, multiple segments in SlideOR can be overlapped and each receieved end-to-end acknowledgement contributes to slide the segments.

In [18], the more MORE protocol has been successfully implemented and tested in realistic simulations with promising results, proving the gain of intra-flow network coding with opportunistic routing in a wireless network.

OMNC [19] is a rate control and routing protocol that aims at improving the end-to-end throughput of lossy wireless networks. OMNC uses multiple paths to push intra-flow coded packets to the destination. In OMNC, the coding and broadcast rate are allocated to transmitters through a distributed optimization algorithm that maximizes the advantage of path

diversity while avoiding congestion. The coding and broadcast rate of each node are matched with its channel status to avoid congestion.

## 2.6 Conclusion

In this work, we focus on linear intra-flow network coding for unicast flows. Intra-flow linear coding has become a well investigated technique for reliable transfer over lossy networks. It showcases two main benefits: it increases transmission reliability by preventing packet losses, and it simplifies retransmissions, because there is no need to retransmit specific pieces from original data.

Intra-flow network coding is particularly suited for wireless networks, especially for the ones relying on opportunistic routing. Indeed, contrary to wired links, wireless links are characterized by high packet loss rates, because packets might be lost during transmission on a link due to interferences or obstacles.

With intra-flow coding, the source node generates random linear combinations for each batch of outgoing packets. To compensate losses, more combinations than original packets may be sent. Network coding allows to send more combinations than original packets.

The destination only needs to get as many combinations as orginal packets to achieve decoding, therefore original data can be recovered even if some packets are lost. The ratio between sent combinations and original packets is called redundancy factor. Adapting the redundancy factor is critical so data is received properly and the overhead is kept low. We believe it should be done by taking into account application requirements. That is what we are going to defend in the next part.

# Part I

# Redundancy adaptation

# Chapter 3

# Dynamic redundancy

## 3.1 Objective and Motivations

One of the benefits of linear intra-flow network coding is to transparently recover from link losses. To recover losses, the system only needs to ensure the destination gets enough innovative combinations as any innovative combination can contribute to the decoding process.

Another main application of network coding is opportunistic routing that has emerged as an interesting approach to resist losses in error-prone environments. In conventional networks, the routing process chooses, for each packet, the next hop before any transmission. However, when the links are lossy, the probability of packet transmission is very low, leading to performance degradation. In contrast, opportunistic routing allows any node receiving packets to participate into forwarding them to the destination.

Self-organized networks, used for instance by the army or by disaster response teams, often have unplanned and variable topology, and are characterized by high packet loss. In such environments, not only intra-flow network coding can perform well, the topology offers also a interesting opportunities for opportunistic relaying. Using opportunistic routing is an additional way to recover from link losses, and the routing can also benefit from intra-flow network coding.

When generating combinations at the source, or regenerating new combinations at nodes, it is critical to make sure that the destination node can solve the system of equations corresponding to the received combinations so that the original data can be decoded. This is possible only if the rank of the system is sufficient, *i.e.*, the destination gets at least the same number of linearly independent combinations than original packets to decode.

Therefore, in lossy environments, nodes need to generate more outgoing coded combinations than incoming packets to mitigate losses. The ratio between outgoing and incoming packets is called redundancy.

Redundancy adaptation is a key issue for network coding. It directly influences application loss, network overhead and congestion. It also indirectly affects, for instance, flow fairness, as we will see in chapter 5. Using fixed code redundancy can lead to a waste of bandwidth in case of over-redundancy or unsuccessful decoding due to an insufficient number of available independent combinations.

In this chapter, after reviewing existing redundancy control mechanisms, we present our first contribution. We derive a simple redundancy bound to set the network coding redundancy

according to the link quality and the targeted maximum application loss rate.

Then, we propose a distributed scheme which allows each data producer to opportunistically make use of multiple available paths in the network to route the coded packets to the destination offering a reliable data delivery with an optimized redundancy control. We restrict the study to only one destination node, even if the work could be extended to multiple receivers.

Finally, we discuss the operating mode of nodes, as they can either send combinations as soon as possible or wait for a complete batch. Indeed, this behavior has a significant impact on performance.

## 3.2 Redundancy control mechanisms

Various redundancy adaptation mechanisms have been proposed. Most of the time, they are introduced as a simple feature of a full-featured network coding scheme. The majority of them rely only on packet losses to adapt the redundancy with different methods.

Redundancy is a parameter of primary concern for network coding schemes in current network systems, especially wireless environments. However, in the literature, most of the works use simplistic methods to evaluate the redundancy, emphasizing on other aspects of their developments.

There are different approaches to adjust code redundancy:

- Static redundancy: the nodes generate combinations with a static redundancy ratio, without considering network environment.
- Adaptive redundancy:
  - Loss rate measurement: nodes estimate the necessary redundancy by measuring packet loss over network links. the sender injects probe packets into the network. Based on the number of received and lost probe packets, the sender can guess the loss rate and adapt the sending rate.
  - Explicit feedback: the receiver provides a direct feedback to the sender, *i.e.*, the number of lost packets. The sender can use the feedback directly or indirectly to adjust the redundancy factor.

### 3.2.1 Static redundancy

The simplest method is to define a static redundancy. The redundancy factor is pre-set according to the expected network characteristics and does not evolve over time. If it is set too high, it generates useless traffic, wasting network resources, and if set too low, the receiver might not be able to recover sent packets.

In the literature, some works have used static redundancy.

In CodeCast [20], network coding is used for intermittent multicast ad hoc networks. Its aim is to show the potential of random linear coding in implementing path diversity and localized loss recovery for multicast video streaming scenarios. CodeCast successfully manages to control packet loss while at the same time keep the latency of multimedia applications in

check, yet it assumes fixed redundancy, which needs to be manually adapted for optimal network performance.

TCP/NC [11] proposes to run a modified TCP stack over sliding-window coding. Its goal is to enhance the TCP flow throughput by protecting TCP from random losses. Indeed, losses are interpreted by TCP as congestion signals and TCP tend to reduces throughput uselessly as a result. In the first version, the redundancy factor is static, and therefore should be tuned manually according to network conditions.

### 3.2.2 Loss rate measurement

The first way to adapt the redundancy is to measure the loss rate. This can be done using probe packets or performing measures on already sent packets.

#### 3.2.2.1 Using probe packets to estimate loss rate

The first family of methods uses probe packets to estimate the loss rate. The sender injects probe packets into the network, then, based on the number of received probe packets, it is able to measure the loss rate and adapt the code redundancy. Another possibility is for the sender to take advantage of previous traffic, since counting lost packets enables to estimate the current loss rate.

In MORE [15] to resist random losses, the authors suggest sending probe packets to predict the loss rate. MORE uses the Expected Transmission Count (ETX) metric to infer the number of combinations a node should forward. The ETX metric for a link is defined as $\frac{1}{1-p}$, where $p$ is the link loss probability [14]. To estimate this probability, nodes rely on probe packets regularly sent over each link.

CodeOR [16] and SlideOR [17] are enhancements of MORE. They use the same mechanism to estimate the loss rate. They have been proposed to enhance MORE in terms of bandwidth utilization.

In short, probing based loss measurement provides a simple solution to locally adapt the redundancy factor, however, it introduces a non negligible overhead and may not reflect the instantaneous link quality due to the changing network conditions.

#### 3.2.2.2 Using sent packets to predict the loss rate

In other existing works, the loss rate is inferred through the number of sent packets, as in the following works: Combo Coding [21] has been developed to enhance the performance of TCP on wireless networks. To this end, it features a mix of intra-flow Pipeline Coding with adaptive redundancy and inter-flow coding of TCP ACKs with TCP DATA. In this scheme, since the integration with TCP is rather tight, code redundancy is adjusted using feedback messages piggybacked by TCP acknowledgments, but to be sure to prevent under-redundancy, the way of adjusting redundancy may possibly lead to over-redundancy situations in most cases.

CodeMP [22] an extended version of Combo Coding for TCP traffic in MANETs. It is also based on Pipeline Coding and works with the same redundancy adaptation scheme, but, contrary to Combo Coding, it features multi-path routing. It that has been shown to perform particularly well in disruptive networks.

In Combo Coding and CodeMP, to estimate per link loss rate, the number of coded packets received in the current generation is piggybacked on TCP ACKs. Provided TCP ACKs are routed on the same path as TCP DATA in the network, and knowing the number of generated packets in the generation recorded locally, each node can obtain the loss rate $p$ as a simple ratio and smoothed with an exponential moving average. Then the redundancy is estimated as:

$$r = (K - 1) + \frac{1}{1 - p} \tag{3.1}$$

where $K$ is the base redundancy that is needed at node i in the absence of losses. $K$ is used to introduce extra redundancy to compensate for local loss rate variations. In the tests, $K = 1.4$, which potentially leads to a redundancy factor set way too high, leading to a huge overhead.

In I$^2$NC [23], a scheme mixing inter-flow and intra-flow network coding, the intra-flow redundancy factor is estimated according to the link loss rate at each hop. The link loss rate is calculated at each intermediate node as one minus the ratio of correctly heard packets in a generation, then broadcasted to neightbors, and averaged with a moving average.

Measurement based on sent packets tends to reflect the links quality better than using probe packets. However, this requires acknowledgements at link level. It's only possible to piggyback on other protocols acknowledgements, like TCP ACKs, if the go through the same links. Besides, this requires elaborate mechanisms to have relays cooperate.

### 3.2.3 Explicit feedback

Other methods to adapt redundancy are based on Automatic Repeat Query (ARQ) mechanisms. They make use of explicit feedback with acknowledgements from the destination. Network coding does not require selective acknowledgements, since the destination only need to inform the source of the number of combinations to generate. The source reacts to the information in two distinct manners:

- It resends enough combinations so that the receiver can decode the current generation.
- It updates its redundancy factor for future data, to allow the receiver to be able to decode next generations without asking for more combinations.

The explicit feedback provides immediate reaction to present loss and adaptation to future loss. It can work between source and destination only (end-to-end) or between each relay node (hop-by-hop).

In the literature, several works rely on explicit feedback to adapt the redundancy. The first version of TCP/NC [11], uses a form of intra-flow coding with static redundancy. Since this is not flexible enough, Adaptive TCP/NC (ATCP/NC) [24] has been introduced afterward.

The main idea of ATCP/NC is to adjust dynamically $r$ to an optimal rate based on the actual network conditions, which can be detected by collecting the feedback information available.

To accomplish this objective, the authors make use of the TCP Vegas loss predictor to detect when congestion is present in the network, in conjunction with the number of duplicate ACKs generated by the receiver. The main goal is to address the inability of the standard TCP protocol to distinguish between congestion losses and random packet losses happening when transmitting over a lossy link.

The authors implement the Vegas Loss Predictor at the NC layer to adjust the redundancy factor R dynamically. The expected throughput is given by $\frac{window}{BaseRTT}$, where $window$ is the difference between the SEQs of the newest and oldest byte involved in the coded packets in transmission and $BaseRTT$ is the $RTT$ of a segment when the connection is not congested.

The measured throughput is given by $\frac{windowAcked}{RTT}$, where $windowAcked$ is the number of bytes ACKed during the last RTT, and $RTT$ is the sample $RTT$ of the received segment. To estimate the cause of the packet losses, the parameter $qv$ is calculated as:

$$qv = (\frac{window}{BaseRTT} - \frac{windowAcked}{RTT})BaseRTT \tag{3.2}$$

Given two thresholds $\alpha$ and $\beta$, if $qv \geq \beta$, the network is too lossy. If $qv \leq \alpha$, random losses, and finally if $\alpha < qv < \beta$, the predictor assumes the network is stable. In pratice, $\alpha = 1$ and $\beta = 3$ are recommended values.

This redundancy control defines a loss differentiation scheme with the help of the Vegas Loss Predictor. Basically, each time an ACK is received, $qv$ is evaluated. If the value is below the threshold, it indicates the transmitting channel is noisy and causes random losses. In these situations, the correct approach is to increase $r$ to mask the losses. Else, $r$ is decreased, allowing TCP to also reduce its rate.

CCACK [25] has been proposed as a scheme for opportunistic routing using intra-flow network coding and featuring an elaborate acknowledgement system. To avoid network congestion, in CCACK, downstream nodes to inform upstream nodes whatever is exactly stored in their buffer in a *Cummulative Coded Acknowledgement*. The CCACK is a compressed version of multiple single ACKs from downstream nodes to upstream nodes. Instead of every ACK transmission to inform the upstream, the most idle downstream node can send the CCACK, its neighbors can overhear and suppress sending the CCACK. This can help to reduce the overhead and provide a good approach to resist the random losses.

In DynCod [26], the authors suggest an intra-flow network coding scheme with end-to-end adaptive redundancy control inspired from TCP/NC. The main idea is for the destination to inform the source about packet losses and about the decodability of the last data sent, via acknowledgement packets. Particularly, the authors change the meaning of TCP ACKs: the destination does not only acknowledge degrees of freedom, but also announces how many unseen packets there are in the coding window at the destination. The principle is that unseen packets in the decoding window at destination can be interpreted as losses on the transmission path. For each packet requested to be sent by TCP, the source will generate one coded packet along with a number of redundant traffic. The number of redundant packets to generate $r$ is calculated according to the number of losses, which can be inferred from the number of reported unseen packets.

To conclude, with explicit feedback, loss rate is not taken into account in determining how many packets are transmitted. Instead, data arrival at the destination is directly informed to the source. If the protocol is not end-to-end, it requires control over acknowledgements routing since information must be piggybacked for use by relays.

### 3.2.4 Conclusion

Figure 3.1 sums up the different redundancy control mechanisms. Adaptive rendundancy can be divided in two categories given the way information is obtained: loss measurement (through probe packets or through sent packets), and explicit feedback. Feedback allows to keep track of the decoding at destination, but the required end-to-end feedback with routing through the same relays make deployment more complex.

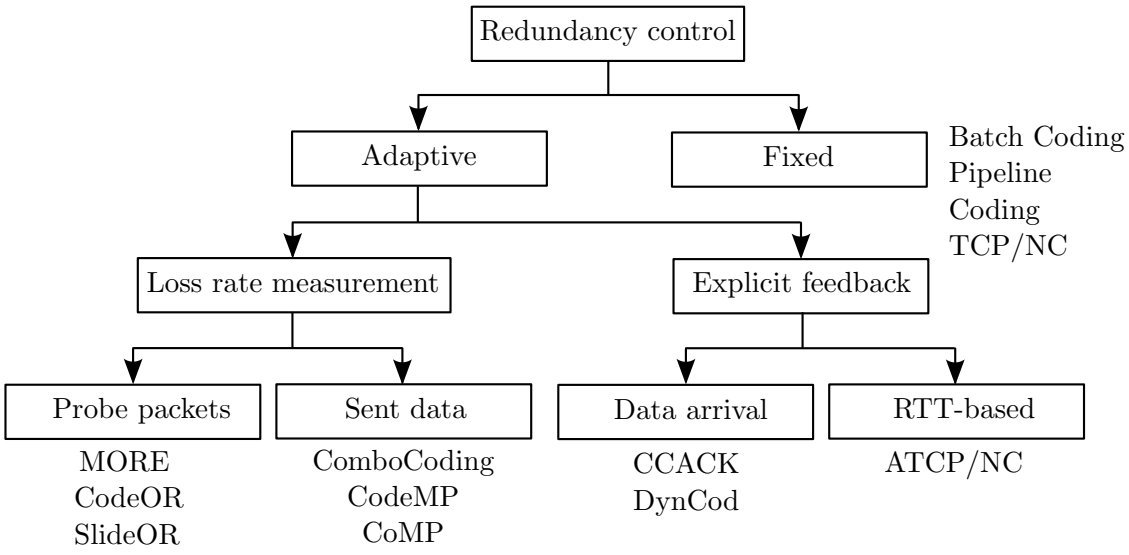



Figure 3.1: Redundancy control for network coding taxonomy

## 3.3 Redundancy estimation

The first aim of our work is to find a proper way to derive the minimum amount of redundancy given the network loss rate without relying on coarse average redundancy values. The redundancy setting should offer a proper guarantee in terms of maximum tolerated application-level loss.

Network coding is performed in batches of $m$ packets, called generations. Redundancy must be adapted in such a way the rate of undecodable generations caused by an insufficient number of received combinations is kept bounded.

If the redundancy factor is too low, the destination could be unable to decode part of the original data. If the redundancy factor is set too high, it leads to a network overhead that impairs performance.

We believe redundancy is actually a compromise between network overhead and application tolerance. Depending on the application, for instance file transfer or video streaming, the tolerance to losses is vastly different.

Therefore, the redundancy adaptation algorithm should not only take into account network characteristics, but also application requirements. When application requirements are available, the redundancy factor should be tuned to be sufficient for the application, but not higher.

### 3.3.1 Generation loss probability bound

In the following section, we are interested in deriving a minimal redundancy to send traffic over a lossy link. We want this redundancy bound to take into account both link loss and application requirements.

We consider a node sending packets to another node over a lossy link with uniform loss rate $p$. The sender, in order to send $m$ packets, applies the redundancy ratio $r$ and transmits $n = \lceil rm \rceil$ combinations.

For the destination to decode the original packets successfully, it is necessary, but not sufficient, that the network coding redundancy compensates link losses on average:

$$(1-p)r \geq 1$$

Moreover, the value of $r$ should not be considerably more than this minimal value, as it should only be a bit more to compensate variations around the mean.

Therefore we have the constraints:

$$r \geq r_{\min}(p) = \frac{1}{1-p} \text{ and } r \simeq r_{\min}(p)$$

Let $X$ be the random variable corresponding to the number of combinations lost during transmission. Then the probability $P(X > (r-1)m)$ of losing more than $(r-1)m$ combinations, *i.e.*, the probability to lose a given generation, follows a cumulative binomial distribution. The Chernoff bound method as stated in B.1 gives the inequality:

$$P(X > (r-1)m) \leqslant \exp\left(-prm(1-\lambda+\lambda\ln\lambda)\right)$$

$$\text{where } \lambda = \tfrac{r-1}{pr}$$

$$\begin{aligned} L(m,p,r) &= \exp\left(-prm(1-\lambda+\lambda\ln\lambda)\right) \\ &= \exp\left(-m\frac{p}{1-p\lambda}(1-\lambda+\lambda\ln\lambda)\right) \end{aligned}$$

Since $r \simeq r_{\min}(p)$, $\lambda \simeq 1$, $\ln(\lambda) = (\lambda-1) - \frac{(\lambda-1)^2}{2} + o((\lambda-1)^2)$, and $1-\lambda+\lambda\ln\lambda \simeq \frac{(\lambda-1)^2}{2}$, we have:

$$\begin{aligned} L(m,p,r) &\simeq \exp\left(-\frac{m}{2}\frac{p}{1-p\lambda}(\lambda-1)^2\right) \\ &= \exp\left(-\frac{1}{2}mpr\left(\frac{r-1}{pr}-1\right)^2\right) \end{aligned}$$

We get the approximated generation loss probability upper bound:

$$P(X > (r-1)m) \leqslant L(m,p,r)$$

$$L(m,p,r) \simeq \exp\left(-\frac{1}{2}mpr\left(\frac{r-1}{pr}-1\right)^2\right) \tag{3.3}$$

This generation loss probability upper bound depends on the redundancy $r$, the loss $p$, and the generation size $m$. As expected, the bound is lower when $m$ is higher and thus a small generation is lost more easily (figure 3.2). Especially for small generation sizes, the redundancy should be higher than the minimal redundancy to guarantee low generation losses.

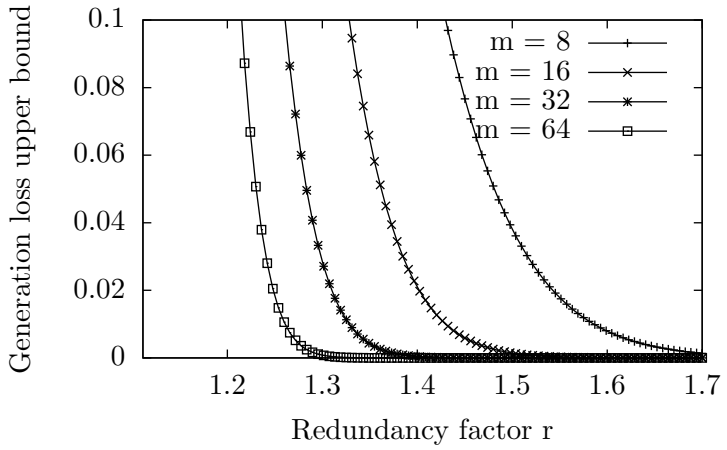



Figure 3.2: Generation loss probability upper bound given the redundancy factor $r$ for different values of generation size $m$ ($p = 0.1$)

### 3.3.2 Redundancy lower bound

Let $\tau$ be the maximum application loss rate. A sufficient condition for the generation loss probability to be less than $\tau$ is $L(m, p, r) \leq \tau$.

$$\exp\left(-\frac{1}{2}mpr\left(\frac{r-1}{pr}-1\right)^2\right) \leq \tau$$

Using the approximation (3.3), we get, after solving the inequation:

$$r \geq \frac{1}{1-p}\left(1 + C\left(1 + \sqrt{1+\frac{2}{C}}\right)\right)$$

$$\text{where } C = \frac{-\ln\tau}{m}\frac{p}{1-p}$$

After the resolution and the approximation detailed in B, given the actual parameters values (in particular $p << 1$), $C << 1$, so $\frac{1}{C} >> 1$ and $1 + \frac{2}{C} \simeq \frac{2}{C}$, and we get the following lower bound for $r$:

$$r \geq r_{\text{bound}}(m, p, \tau) = \frac{1}{1-p}\left(\frac{1 + (1+\sqrt{2C})^2}{2}\right) \tag{3.4}$$

Note if $\sqrt{\frac{-\ln\tau}{m}} << 1$ (*e.g.* $m = 64$ and $\tau \simeq 0.01$), as $(1+\sqrt{2C})^2 = 1 + 2\sqrt{2C} + o(\sqrt{2C})$, we can use the following approximation:

$$r \geq r_{\text{bound}}(m, p, \tau) \simeq \frac{1}{1-p}\left(1 + \sqrt{2C}\right) \tag{3.5}$$

The redundancy lower bound can be calculated given a generation size $m$, a loss probability $p$ , and the targeted maximum application loss rate $\tau$.

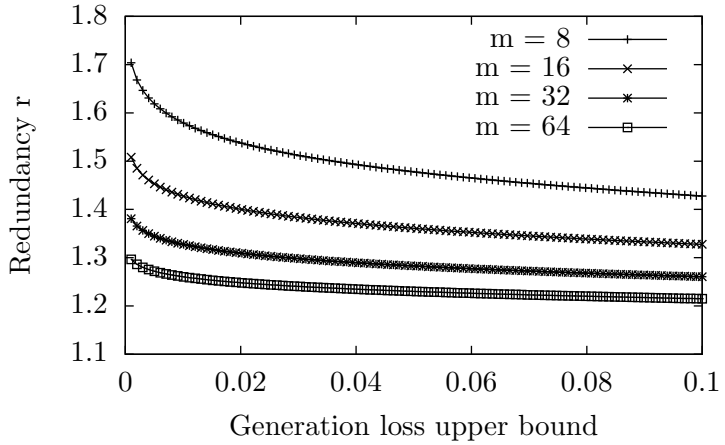



Figure 3.3: Redundancy factor $r_{\text{bound}}$ given the generation loss probability upper bound $\tau$ for different values of generation size $m$ ($p = 0.1$)

### 3.3.3 Non-innovative combinations leading to intrinsic loss

When using random linear coding, combinations are generated with random coefficients. Therefore, it might happen that several generated non redundant combinations are actually non-innovative, *i.e.*, linearly dependent. When this happens, the generated combination might be useless at reception and discarded.

The common method to solve this issue is to keep track of generated combinations and prevent the generation of non-innovative combinations by checking linear dependency of each new combination. However, this technique, besides being more costly in terms of resources, is not applicable with opportunistic routing and multiple relays because a relay could not realistically keep track of all combinations generated by its neighbors in order to check for linear dependency.

Here we consider this phenomenon as an additional source of loss to be corrected through redundancy. To estimate the equivalent loss, let us focus on the decoder receiving a new combination: the situation where it is the most probable that the combination turns to be linearly dependent happens when the rank is $m - 1$ and only one more combination is missing, *i.e.*, the decoder already processed $m - 1$ combinations. Each of them can be used to eliminate one component from the incoming combination, and it is linearly dependent if and only if this also zeros the last component, which happens with probability $p_{\text{int}} = \frac{1}{|GF(2^8)|-1}$ since coefficients are nonzero and uniformly distributed.

Therefore, to compensate for generated non-innovative combinations, we add $p_{\text{int}} = \frac{1}{255} \simeq 0.004$ to the loss rate.

## 3.4 Distributed adaptive redundancy control

Now that we defined a redundancy bound for a link, we want to derivate a distributed algorithm. Deploying a centralized algorithm to adapt redundancy in a whole network is more straightforward but prevents scaling and practical deployment.

In this section, we aim at defining a distributed intra-flow network coding scheme with adaptive redundancy control for reliable data delivery.

The scheme requires no signaling and relies on opportunistic routing: each node can overhear coded packets sent by its neighbors and thus can participate in forwarding the coded packets to the destination. The redundancy bound previously defined allows optimizing the number of redundant combinations a node has to generate towards its neighbors.

More precisely, when a node $i$ overhears a combination sent by a node $k$ to final destination $j$ (figure 3.4), it executes the following steps:

- Node $i$ decides if it should forward the traffic. To do so, it compares its distance from $j$ to the distance of $k$ from $j$. If $k$ is nearest, the packet is dropped, else, the packet is buffered with the already received packets of the same flow and proceeds to the second step.
- Node $i$ computes the necessary redundancy for this flow using link transmission qualities of its neighbors and the redundancy estimation presented above. Finally, $i$ generates coded packets according to $r$ and sends them.

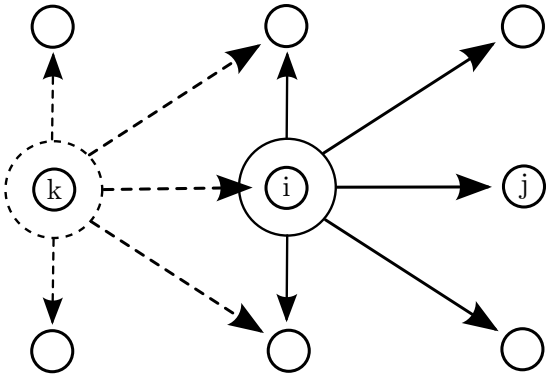



Figure 3.4: Node $i$ overhears a combination from $k$ and sends a new combination.

### 3.4.1 Network model

We model the network as an oriented graph: node $i$ has a link to node $j$ if and only if node $j$ can receive packets from node $i$ with a non-negligible probability.

We assume that every node $i$ has knowledge of the transmission probabilities in the whole network.

We expect this information to be available through a Link-State routing protocol like OSPF or OLSR. Indeed, link transmission probabilities can easily be measured and piggybacked on the routing protocol's Link State Update messages. However, this is a strong assumption since transmission probabilities can change rather often so the resulting signaling traffic could be

unbearable in pratice. This particular issue is tackled in the enhancement proposed in the next chapter .

We use the following notations:

- $m$: Network coding generation size
- $\tau$: Maximum tolerated application loss for the flow
- $V_i$: Set of neighbors of node $i$, *i.e.* nodes which can receive packets from $i$
- $d_{ij}$: Distance in hops between nodes $i$ and $j$ in the graph
- $p_{ij}$: Link loss probability, *i.e.* loss probability for a packet transmission from $i$ to $j$
- $q_{ij}$: Link transmission probability, as measured by node $j$ ($q_{ij} = 1 - p_{ij}$)

For any pair of nodes $(i, j)$, $j \in V_i$ if and only if $q_{ij} > \varepsilon$, where $\varepsilon << 1$ is a link transmission probability under which the link contribution is considered negligible.

Finally, let $Q = [q_{ij}]_{i,j \leq N}$ be the transmission quality matrix at 1-hop distance, *i.e.*, the probability of hearing a packet between neighbors.

$$Q = \begin{pmatrix} 0 & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & 0 & \cdots & q_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ q_{N,1} & q_{N,2} & \cdots & 0 \end{pmatrix}$$

### 3.4.2 Algorithm at relays

As mentioned previously, upon overhearing from $k$ a combination to destination $j$, $i$ runs the following algorithm:

#### 3.4.2.1 Forward or drop

Node $i$ checks if it is one of the next hops, that is if no strictly shorter path exists from $k$ to $j$, *i.e.*, if:

$$(Q^{d(i,j)-1})_{kj} < \varepsilon \text{ with } \varepsilon << 1$$

If $i$ is a next hop, it proceeds to the next step, else the packet is dropped.

#### 3.4.2.2 Redundancy estimation

We expect the network to transmit $\sigma = \sum_{v \in N_{kj}} q_{iv}$ combinations for each combination sent by $k$, where $N_{kj}$ is the set of neighbors from $k$ to $j$ computed by $i$:

$$N_{kj} = \{v \in V_k / (Q^{d_{kj}-1})_{vj} \geq \varepsilon\}$$

$i$ has to send on average $\frac{1}{\sigma}$ combinations for each combination sent by $k$.

However, we want enough innovative combinations to go through, this means that every combination should be received by at least one of the neighbors of $i$. If it is not the case, too few independent combinations would be forwarded by the network and the destination would be unable to decode.

The probability of one combination to be received by none of the neighbors, *i.e.*, lost, is $\prod_{v \in N_{ij}} 1 - q_{iv}$, where $N_{ij}$ is the subset of neighbors for destination $j$:

$$N_{ij} = \{v \in V_i / (Q^{d_{ij}-1})_{vj} \geq \varepsilon\}$$

Therefore, using the redundancy bound previously defined, we get:

$$r = \frac{1}{\sigma} r_{\text{bound}}(m, \prod_{v \in N_{ij}} 1 - q_{iv}, \tau')$$

We set $\tau'$ as the goal link loss in order to guarantee the end-to-end loss $\tau$, which means it has to verify $(1-\tau')^{d_{sj}} \geqslant 1 - \tau$, *i.e.*, $\tau' \leqslant 1 - (1-\tau)^{\frac{1}{d_{ij}}}$. Since $d_{sj} \leqslant d_{si} + d_{ij}$, where $s$ is the source node, we use:

$$\tau' = 1 - (1-\tau)^{\frac{1}{d_{si}+d_{ij}}} \simeq \frac{\tau}{d_{si} + d_{ij}}$$

Eventually, $i$ has to send with the redundancy:

$$r = \frac{1}{\sigma} r_{\text{bound}}(m, \prod_{v \in N_{ij}} 1 - q_{iv}, \frac{\tau}{d_{si} + d_{ij}})$$

Note that $r$ allows to determine the number of redundant packets to generate, at the same time, it aims at preventing useless redundancy while keeping the advantage of multipath in forwarding the packets to the destination.

The $r$ value, calculated in a node, could be less than 1 in certain situations where the number of neighbors is large. In this case, a node may not add redundant packets locally, however, the optimized number of redundant packets is achieved globally through the neighboring nodes.

### 3.4.3 Redundancy at the source

The above coding redundancy estimation is applicable for network nodes. If a node $i$ is the source of a flow, it executes only the second step with $\sigma = 1$, so $i$ has to send with the redundancy:

$$r = r_{\text{bound}}(m, \prod_{v \in N_{ij}} 1 - q_{iv}, \frac{\tau}{d_{si} + d_{ij}})$$

### 3.4.4 On-the-fly recoding vs delayed recoding

In several schemes based on Pipeline coding [10] [22], relays use *on-the-fly recoding*.

With *on-the-fly recoding*, when a relay gets an innovative combination from a neighbor, it adds the combination to locally cached packets of the same flow and generation, and new combinations are generated and sent immediately. This technique can help reduce transmission

delays, however, it comes at the cost of a major drawback, as it strongly increases the risk of generating non-innovative combinations.

By comparison, with *delayed recoding*, when a node receives an innovative combination, it stores it, but it does not immediately generate new combinations. Instead, it recodes new combinations with redundancy $r$ only if the generation is complete or if no other combination is following for a short while.

We use this approach in our tests to generate less non-innovative combinations and therefore increase the effectiveness of network coding.

## 3.5 Simulation results

In the following sections, we evaluate our model through comprehensive simulation analysis. To this end we have developed a custom program in C++ that implements the proposed algorithm. The program simulates packet transmissions, link loss, coding and decoding operations.

### 3.5.1 Loss bound evaluation

The first step of our evaluation consists in verifying the generation loss rate bound we derived in the first part of the paper. The setup is simple: we consider a coding node sending combinations to a decoding node through a link with uniform loss $p$, with generation size $m$ and redundancy $r$, and we measure the generation loss rate on the decoder side.

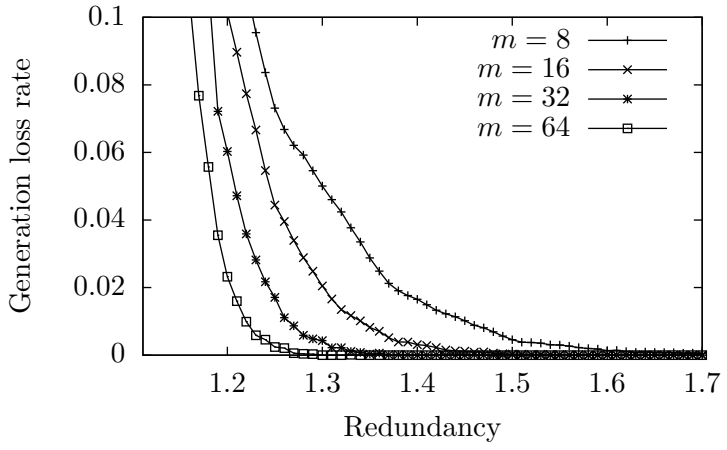



Figure 3.5: Measured generation loss probability given the redundancy factor $r$ for different values of generation size $m$ ($p = 0.1$)

Figure 3.5 shows clearly that the average redundancy designed to compensate for link loss, here $r_{\min} = \frac{1}{1-p} \simeq 1.11$, is not adequate in practice since it would lead to a generation loss even higher than link loss $p$, thus defying the purpose of network coding to protect from link errors.

By comparing figure 3.5 with figure 3.2, one easily verifies that measured generation loss rates are below the upper bounds. However, in this case, with a rather high value of $p$, the bound tends to be too loose for small generation sizes. The bound is tighter for lower loss $p$ and for larger generation size.

### 3.5.2 Distributed adaptive redundancy evaluation

We evaluate the redundancy estimation algorithm against three schemes (see bellow) in three different corridor topologies, with respectively one, two and three parallel lines of relays (figure 3.6). Transmissions between relays experience uniform link loss $p$ to simulate interference in the network.

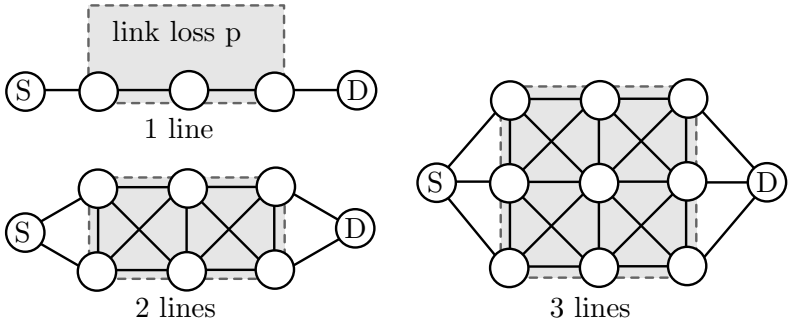



Figure 3.6: Topologies with different numbers of lines of relays with link loss $p$ between relays

Two parameters are measured: loss at destination, and relative overhead which is calculated as follows:

$$\text{relative overhead} = \frac{\#\text{ packets sent}}{\#\text{ nodes} \times \#\text{ original packets}}$$

We run transmissions from source to destination using generation size $m = 32$ with our redundancy adaptation scheme ($\tau = 0.01$), and the following schemes:

- *Average*, where each node sends with redundancy $\frac{1}{1-p}$
- *Static*, where each node sends with a predefined constant redundancy factor $r$
- *CodeMP-like*, where each node sends with redundancy $\frac{1}{1-p} + K$. The $K$ constant is predefined and needs to be set high enough for this scheme to work, however a too large value leads to over-redundancy situations. CodeMP [22] uses this scheme with $K = 0.8$, which produces an extremely high redundancy.

With only one line of relays, only our scheme achieves near-zero loss at destination (figure 3.7), however, this requires a more overhead than the other schemes (figure 3.8). As expected, the average redundancy performs badly even with low link losses along the path, contrary to the other schemes using added static redundancy.

With more than one line of relays, overhearing and opportunistic relaying is sufficient for all the schemes to achieve near-zero loss at destination. However, the overhead is much lower with our algorithm, since it doesn't greedily use network resources (figure 3.9). This is even more visible with three lines of relays (figure 3.10).

### 3.5.3 Resilience to coding/no coding nodes

A given node in the network could choose to simply forward packet without coding. In such a case, the node must remember the previously forwarded packet sequence numbers in order to

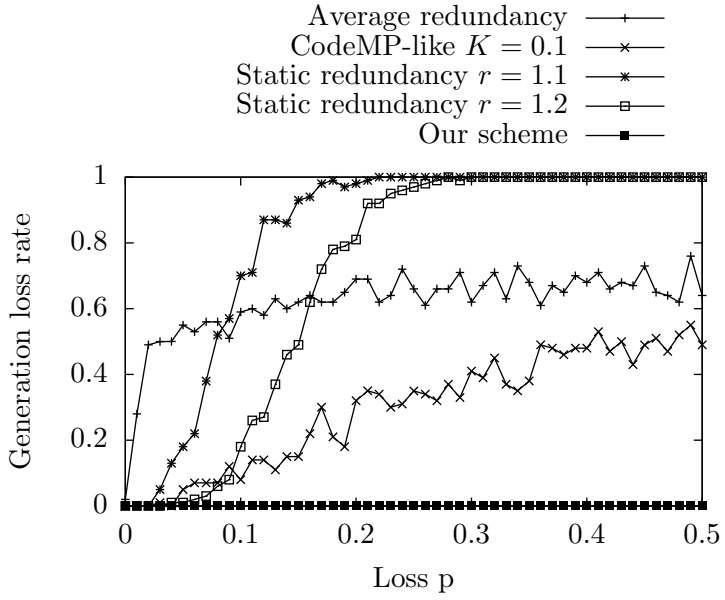



Figure 3.7: Generation loss at destination given relay loss $p$ with one line of relay nodes

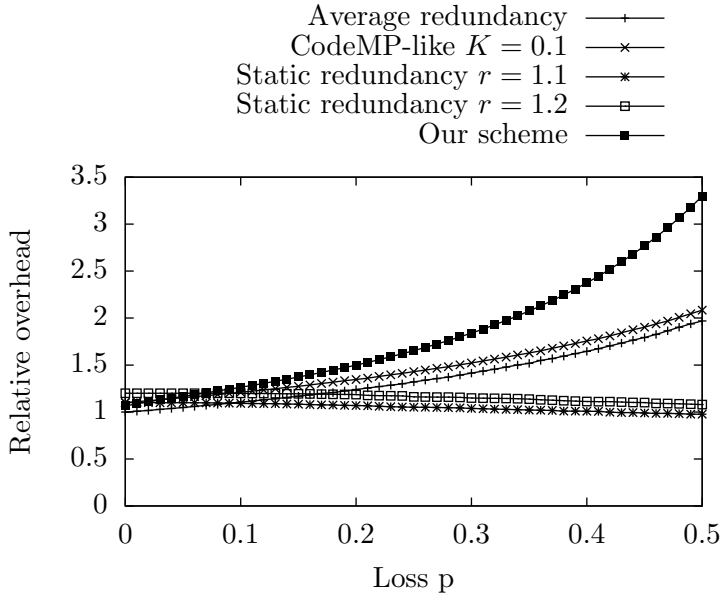



Figure 3.8: Relative overhead given relay loss $p$ with one line of relay nodes

suppress duplicates. A node does not code for many possible reasons, for example, when a relay does not have adequate processing power or sufficient battery reserve, or cannot be trusted.

Forwarding saves resources, but has two drawbacks: it does not allow to balance redundancy, and it degrades performance if too few nodes recode. The problem arises in situations where multiple upstream paths with only non recoding nodes exist for the same downstream node.

Figure 3.12 shows the effect on overhead of non coding nodes. The results are counter-intuitive. At first, the more non coding nodes, the more network overhead then after a plateau, the overhead decreases again. Note we use the reference value of 1 for network load in the

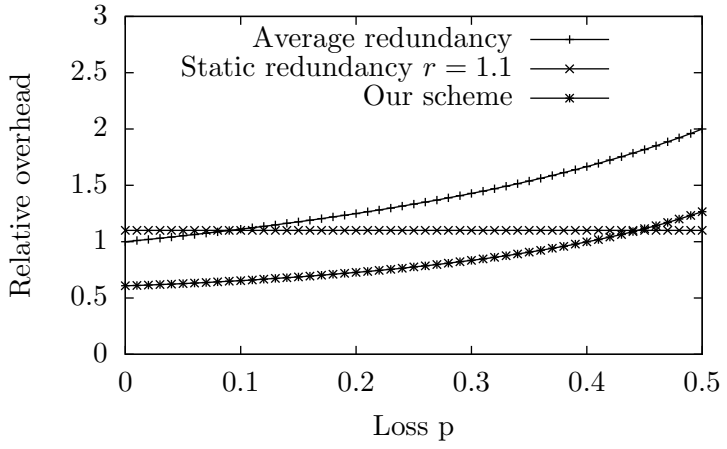



Figure 3.9: Relative overhead given network relay loss $p$ with two lines of relay nodes

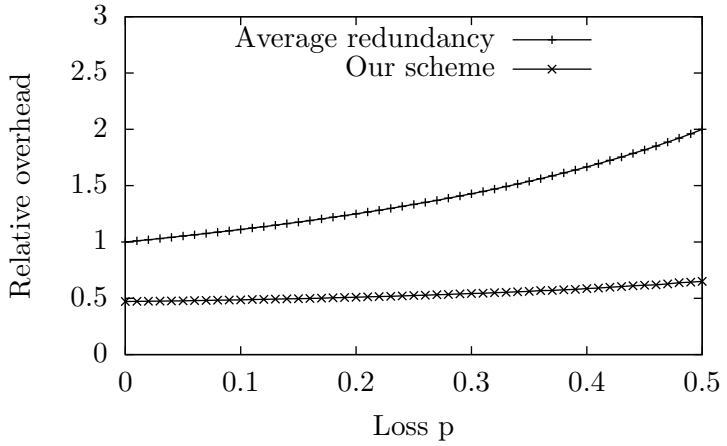



Figure 3.10: Relative overhead given relay loss $p$ with three lines of relay nodes

hypothetic conditions where there is no link loss and every node is in forward-only mode and transmits opportunistically every packet it overhears.

This is due to the behavior of forwarding (*i.e.*, non recoding) nodes: they are able to drop duplicate packets that have already been routed once, but they are unable to detect innovative combinations, so they forward everything without discrimination. This means that initially, as the number of forwarding agents increases, the redundancy added by coding nodes on the links floods the network even more than it would do without coding nodes.

### 3.5.4 Impact of delayed recoding

On-the-fly recoding, where new combinations are generated upon receiving an innovative combination, is a common mechanism motivated by the need to reduce delays. This practice, however, tends to dramatically hurt performance as shown in the experiment below.

Our results show that, even in the 1-line topology (figure 3.13), on-the-fly recoding performs very poorly compared to delayed recoding, especially with growing number of relays.

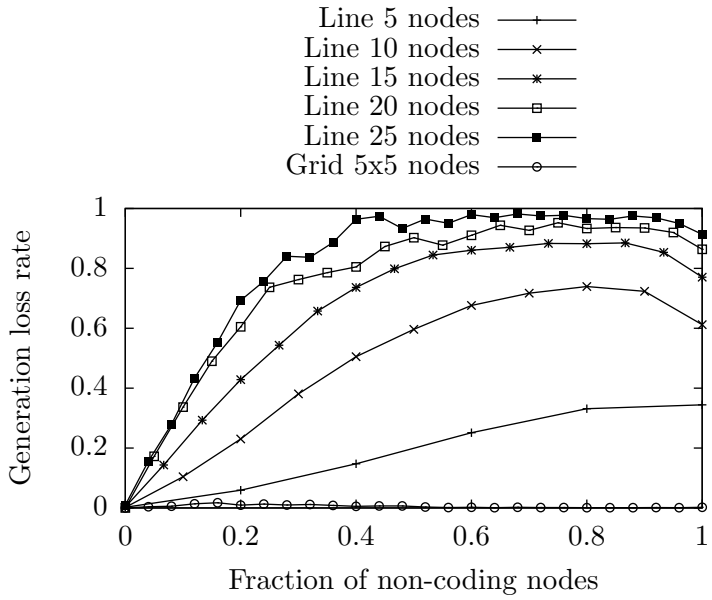



Figure 3.11: Generation loss at destination in different topologies given the fraction of non-coding nodes

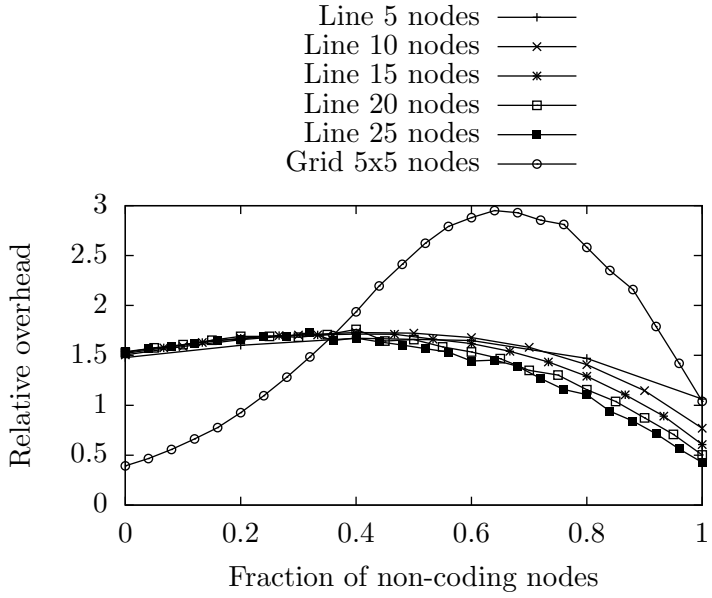



Figure 3.12: Network overhead in different topologies given the fraction of non-coding nodes

Redundancy is exactly the same in the two cases.

As expected, delayed recoding attains less than the required tolerated loss after decoding $\tau = 0.1\%$. In contrast, on-the-fly recoding is far off the mark, with the gap growing with the number of relays, since every hop tends to lose more information by generating suboptimal combinations.

Using on-the-fly coding can reduce delays in certain cases, for instance [10], however, it is inefficient and forces to transmit more coded packets to get the same result, therefore leading

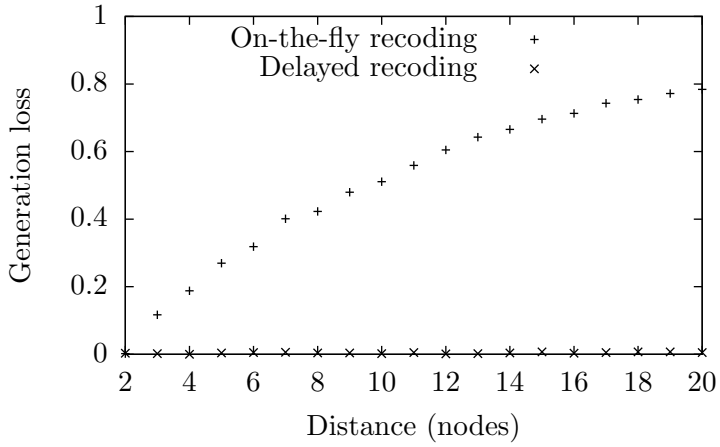



Figure 3.13: Measured generation loss probability for on-the-fly and delayed recoding given the number of hops on the path between source and destination ($p = 0.1, \tau = 0.1\%$)

to more coding overhead.

## 3.6 Conclusion

In this chapter, we explained that in our opinion, network coding redundancy is dictated by the tradeoff between application performance requirements and network overhead introduced by coding redundancy.

To address this problem, we first presented a redundancy bound taking into account, not only link loss rate, but also an target application-tolerated loss. Then, we have proposed a practically-deployable redundancy adaptation algorithm, targeted to wireless meshes, that guarantees a minimum decoding ratio at destination, while keeping the overhead relatively low.

The evaluations we have conducted show that our scheme outperforms static, average and CodeMP-like adaption schemes.

In the next chapter, we will extent ths model to tackle the issue of nodes having different capacities to perform network coding operations. We will also reduce the requirements in terms of knowledge of the network graph, since knowing the near-realtime link qualities for the whole network can be unfeasible in pratice du to the necessary signaling traffic.

# Chapter 4

# Taking constraints into account

## 4.1 Introduction

Network coding implies a non-negligible cost in terms of computing, storage and power consumption. Especially if the network coding is, for instance, protected against pollution by cryptographic mechanisms, like homomorphic signature schemes, the processing cost can be increased by 100 times compared to network coding without protection [27].

Choosing which nodes to use to perform coding operations is a central issue when deploying network coding in wireless mesh networks. It consists in choosing the nodes which perform coding in the network graph, and how much they contribute to it by generating new combinations.

In practice, nodes are typically more or less constrained depending on their physical capabilities and their current status, *e.g.* battery powered or plugged into the mains, the flow should be optimized while enforcing constraints to reduce computing cost especially on the less able nodes.

In wireless mesh networks, nodes often have strong constraints regarding energy consumption and processing power. In such a case, coding and transmitting might represent costly operations. One way to reduce coding costs is to distribute coding operations on a limited number of nodes while trying to optimize specific performance gains. It could be interesting to assign coding operations in priority to nodes with the less constraints, *e.g.,* the less battery-constrained nodes.

## 4.2 Taking constraints into account

Self-organized field networks, used for instance by the army or by disaster response teams, often have unplanned topology and significant device diversity. For instance, some devices can be handheld, and others attached to cars, leading to widely different hardware capacities. Moreover, implementing network coding rather than simply forwarding packets comes at a cost in terms of battery power. Nodes in reality also have widely different available battery capacities: handled devices have limited battery capacity, whereas car devices can have virtually unlimited battery capacity.

To address this challenge, our aim is to propose a constraint awareness distributed algorithm which takes into account the capacity of each network node to contribute to the encoding operations. Each node defines its relative contribution to the network coded transmission depending

on its own constraints. This preference is taken into account when forwarding the packet combinations through the network nodes, while guaranteeing decoding at destination and providing a reliable data delivery.

The remainder of this chapter is organized as follows. First, we provide a selection of existing work related to our proposal. Then, we provide details on our distributed constraint-aware network coding algorithm. Finally, we evaluate the performance of our algorithm through simulation and analyse its effect on network traffic distribution and coding redundancy overhead.

## 4.3 State of the art

Choosing which nodes to use for coding operations is an interesting issue when deploying network coding in wireless mesh networks. It consists of selecting the nodes which perform coding in the network, and how much they contribute to it by generating new combinations. In practice, nodes are typically more or less constrained depending on their physical capabilities and their current status, *e.g.*, battery powered or plugged into the mains. The flow should be forwarded across the network while enforcing constraints to reduce computing cost especially on the less capable nodes.

The authors of [28] suggest to choose specific nodes of the network to perform the coding operations rather than having every intermediate node performing it. To this end, they propose a modified version of Ford-Fulkerson algorithm to maximize the multicast flow in the network graph. The authors address the problem for multicast traffic with the goal of enhancing network capacity through inter-flow network coding. In our work, we consider intra-flow network coding for unicast traffic with the aim of providing reliability transmission.

The authors of [29] propose a social reputation based system to provide network coding incentives in MANETs. They consider three different possible behaviors for nodes, *i.e.* network coded forwarding, simple forwarding and dropping, and they prove the convergence of the described incentive system using games theory.

The INPAC scheme [30] [31] aims at being more practical. It is an incentive scheme for network coding in wireless mesh networks whose convergence has been proven by games theory. Based on MORE [15], it is designed to be deployed on real networks.

In the above schemes, the authors consider selfish nodes which need incentives to participate in the encoding process. In our work, we define collaborative nodes: each node indicates its relative contribution to the network coded transmission and according to the nodes capabilities the coding redundancy is adapted to ensure decoding at destination.

## 4.4 Constraint-aware Network Coding Model

We consider a unicast network-coded flow running over a wireless mesh network with one source node and one sink node, called destination in the following. All the other nodes can serve as opportunistic forwarders: for a given flow, each node can overhear coded packets sent by its neighbors and thus can participate in recombining received combinations of the flow and in forwarding them closer to the destination.

Additionally, each node is characterized by a contribution factor $\alpha \in ]0, 1]$ which indicates its relative contribution to the network coded transmission. The nodes set their factor themselves

depending on their own constraints. The flow's combinations are forwarded in the network with the constraint of decoding at destination, while keeping each node's contribution relatively to its neighbors proportional to its $\alpha$ factor.

For instance, a node with an $\alpha$ factor ten times higher than its neighbors will tend to generate and send ten times more combinations. This mechanism allows to have the combinations generated and sent in priority by nodes possessing the actual capabilities of doing so (Fig. 4.1).

Note that, each node, to ensure decoding at destination, adapts dynamically the amount of redundant packets according to the application loss rate requirements.

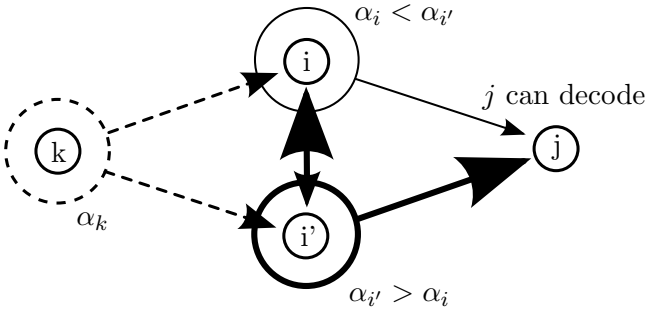



Figure 4.1: Nodes $i$ and $i'$ relay traffic with different contributions.

### 4.4.1 General principle

Each node performing network coding operations can overhear combinations sent by its neighbors. When node $i$ overhears an innovative combination from $k$ for a flow with destination $j$ (Fig. 3.4), the algorithm consists in different steps:

- Node $i$ decides if it should forward the traffic. To do so, it compares its distance, (*i.e.*, hop count) from $j$ to the distance of $k$ from $j$. If $k$ is nearest, the packet is dropped.
- Node $i$ checks if the received combination is innovative. If so, the packet is buffered with the already received packets of the same flow and $i$ proceeds to the next step, else the packet is dropped.
- Node $i$ estimates the actual contribution for this flow according to its contribution factors $\alpha_i$ and link transmission qualities of its neighbors.
- Node $i$ computes the redundancy it has to add to this flow to guarantee decoding at destination.
- Finally, node $i$ generates coded packets according to its contribution $\alpha_i$ and the redundancy calculated as described in the following, and sends them.

### 4.4.2 Redundancy Selection Algorithm

Similarly to the previous chapter 3.4.1, we model the network as an oriented graph and we use the same notations.

- $m$: Network coding generation size

- $\tau$: Maximum tolerated application loss for the flow
- $V_i$: Set of neighbors of node $i$, *i.e.* nodes which can receive packets from $i$
- $d_{ij}$: Distance in hops between nodes $i$ and $j$ in the graph
- $p_{ij}$: Link loss probability, *i.e.* loss probability for a packet transmission from $i$ to $j$
- $q_{ij}$: Link transmission probability, as measured by node $j$ ($q_{ij} = 1 - p_{ij}$)

Moreover, we define the following:

- $A$: Adjancency matrix of the network graph, *i.e.* $A_{ij} = 1$ if and only if $j \in V_i$
- $\alpha_i$: Contribution factor of node $i$, $0 < \alpha_i \leq 1$

Contrary to the model in the previous chapter, we assume that every node has knowledge of the global adjacency matrix $A$, and so of the distances $d_{ij}$, but knows the transmission probabilities $q_{ij}$ and contribution factors $\alpha_i$ only for its neighbors at two hops (*i.e.*, the transmission probabilities between its neighbors and their neighbors, and the contribution factors for its neighbors and for their neighbors).

It is reasonable to assume this information is available through the Link-State routing protocol, since link transmission probabilities can easily be piggybacked on the routing protocol's Hello messages in practice.

### 4.4.3 Algorithm at relays

Upon overhearing a combination from $k$, $i$ runs the algorithm detailed below. We note the flow destination $j$ and the flow source $s$.

**Checking if it is a next hop** First, node $i$ checks if it is one of the potential next hops, that is if no strictly shorter path exists from $k$ to $j$, *i.e.*, if:

$$(A^{d_{ij}-1})_{kj} = 0$$

If $i$ is not a potential next hop, it continues to the next step. If not, the packet is dropped.

**Checking if the combination is innovative** $i$ then check if the combination is innovative, *i.e.*, if it is linearly independent of the combinations it has already received and buffered. If it is, $i$ continues to the next step. If not, the packet is dropped.

**Estimating the actual contribution** Then, node $i$ estimates how many packets it has to generate for each incoming combination, compared with the other next hops.

For each combination sent by $k$, each neighbor $v$ will receive it with probability $q_{kv}$, and its contribution will be weighted by the factor $\alpha_v$, meaning the non-normalized contribution of $v$ is $\alpha_v q_{kv}$. The normalization factor for combinations from $k$ to $j$ is:

$$\sigma = \sum_{v \in N_{kj}} \alpha_v q_{kv}$$

where $N_{kj}$ is the subset of next hops in $V_k$ for destination $j$, *i.e.* neighbors of $k$ that are one step closer to node $j$:

$$N_{kj} = \{v \in V_k / (A^{d_{kj}-1})_{vj} = 1\}$$

Therefore, for each combination sent by $k$, $i$ will receive it with probability $q_{ki}$, and for each received combination it has to generate on average $\frac{\alpha_i}{\sigma}$ combinations.

**Computing the redundancy** We want enough innovative combinations to go through the network. For this reason, every combination should be received by at least one next hops of $i$ which will produce new combinations as a result. If it is not the case, too few independent combinations would be forwarded across the network and the destination might be unable to decode.

The probability of one combination to be received by one neighbor $v$ is $q_{iv}$, so if each neighbor had equal contributions, the probability not to be received and transmitted further by any next hop would be:

$$P(N_{ij}) = \prod_{v \in N_{ij}} (1 - q_{iv})$$

where $N_{ij}$ is the subset of next hops in $V_i$ for destination $j$, *i.e.*, neighbors of $i$ that are one step closer to node $j$:

$$N_{ij} = \{v \in V_i / (A^{d_{ij}-1})_{vj} = 1\}$$

However, nodes will have different contributions when generating new combinations. By definition of alpha, we express the contribution of each node $v$ relative to its neighbors as the ratio between the contribution of the node and the maximum contribution of neighbor nodes $\frac{\alpha_v q_{iv}}{\max_{u \in N_{ij}} \alpha_u q_{iu}}$, then the probability becomes:

$$\begin{aligned} P(N_{ij}) &= \prod_{v \in N_{ij}} (1 - \frac{\alpha_v q_{iv}}{\max_{u \in N_{ij}} \alpha_u q_{iu}} \times q_{iv}) \\ &= \prod_{v \in N_{ij}} (1 - \frac{\alpha_v q_{iv}^2}{\max_{u \in N_{ij}} \alpha_u q_{iu}}) \end{aligned}$$

In chapter 3 and as detailed in appendix B, we derived $r_{\text{bound}}(m, p, \tau)$ as a minimal redundancy, for generation size $m$ and network loss probability $p$, to guarantee decoding at reception with the maximum application-level loss $\tau$:

$$r_{\text{bound}}(m, p, \tau) = \frac{1}{1-p} \left( 1 + \sqrt{\frac{-2 \ln \tau}{m} \frac{p}{1-p}} \right)$$

Therefore, in our context, the minimal redundancy becomes:

$$r_{\text{bound}}(m, P(N_{ij}), \tau')$$

We set $\tau'$ as the goal link loss in order to guarantee the end-to-end loss $\tau$, which means it has to verify $(1-\tau')^{d_{sj}} \geqslant 1 - \tau$, *i.e.*, $\tau' \leqslant 1 - (1-\tau)^{\frac{1}{d_{ij}}}$. Since $d_{sj} \leqslant d_{si} + d_{ij}$, where $s$ is the source node, we use:

$$\tau' = 1 - (1-\tau)^{\frac{1}{d_{si}+d_{ij}}} \simeq \frac{\tau}{d_{si} + d_{ij}}$$

**Generating combinations** Eventually, for each incoming combination of this flow, $i$ has to generate and send on average $r$ combinations:

$$r = \frac{\alpha_i}{\sigma} r_{\text{bound}}\Big(m, \prod_{v \in N_{ij}} (1 - \frac{\alpha_v q_{iv}^2}{\max_{u \in N_{ij}} \alpha_u q_{iu}}), \frac{\tau}{d_{si} + d_{ij}}\Big)$$

$$\text{where } \sigma = \sum_{v \in N_{kj}} \alpha_v q_{kv}$$

### 4.4.4 Special case of the source

If node $i$ is the source of the flow, it generates combinations with $\sigma = 1$ and without taking $\alpha_i$ into account, so it has to send, for every original packet, $r$ combinations:

$$r = r_{\text{bound}}\Big(m, \prod_{v \in N_{ij}} (1 - \frac{\alpha_v q_{iv}^2}{\max_{u \in N_{ij}} \alpha_u q_{iu}}), \frac{\tau}{d_{si} + d_{ij}}\Big)$$

### 4.4.5 Special case of neighbors of the destination

Neighbors of the destination $j$ take $\alpha_j = 1$ even if $j$ announced a different $\alpha$.

## 4.5 Simulation results

In the following sections, we evaluate our model through comprehensive simulation analysis. To this end we have developed a custom program in C++ that implements the proposed algorithm. The program simulates packet transmissions, uniform link loss, coding and decoding operations. The application maximum loss is set to $\tau = 0.01$ and the link loss to $p = 0.1$.

### 4.5.1 Effect of the contribution factor

The contribution factor $\alpha$ allows to balance traffic between a node and its neighbors. We showcase the effect in a topology with multiple lines (Fig. 4.2) by reducing $\alpha$ on the nodes in the first line.

Other things equal, when multiple nodes overhear packets from the same flow, their respective relative contribution will be proprotionnal to their $\alpha$ factor. For instance, a next hop with $\alpha = 0.5$ will tend to generate half the number of combinations than a next hop with $\alpha = 1$, thus allowing it to save some power if necessary.

As expected, lowering the $\alpha$ factor on the first line has the effect of diverting the traffic to the other lines where the nodes have $\alpha$ set to 1, in particular to the second one (Fig. 4.3).

We measure the overhead relatively, with the value of 1 corresponding to network flooding, *i.e.* if every node forwards every combination once. Therefore, a value less than 1 shows the scheme prevents useless flooding even when guaranteeing decoding at destination.

Changing the $\alpha$ factor has a limited cost in terms of overhead, since the traffic is mainly only moved across the network (Fig. 4.4) as nodes with higher contribution factors generate a bigger part of the combinations.

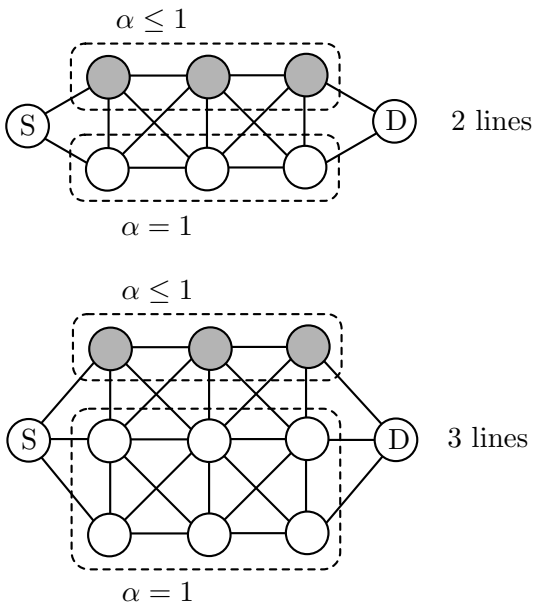



Figure 4.2: Simulated network topologies with 2 or 3 lines

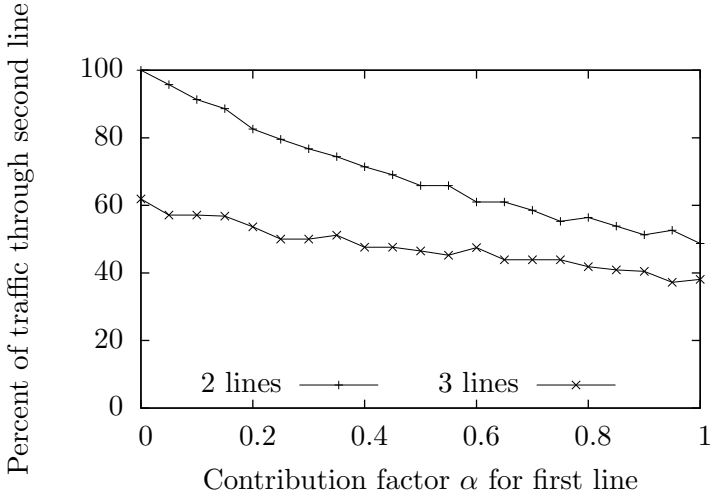



Figure 4.3: Effect of $\alpha$ factor on first line

The slight increase of network overhead for lower values of $\alpha$ for the first line comes from the fact that less possible paths are available. Indeed, redundancy is added to be sure every independent combination has been received and processed by at least one next hop, therefore, less available next hops means more redundancy has to be added to keep the same guarantee and have enough independent combinations at destination, leading to more overhead.

### 4.5.2 Impact of nodes with lower contribution factors

The induced network overhead is expected to raise with the number of nodes with a low $\alpha$ factor in the network. We measure it with randomized patterns of nodes with $\alpha = 0.1$ among nodes with $\alpha = 1$ in a variable-size grid setup (Fig. 4.5).

As with the previous overhead measures, we notice the overhead does not vary a lot depend-

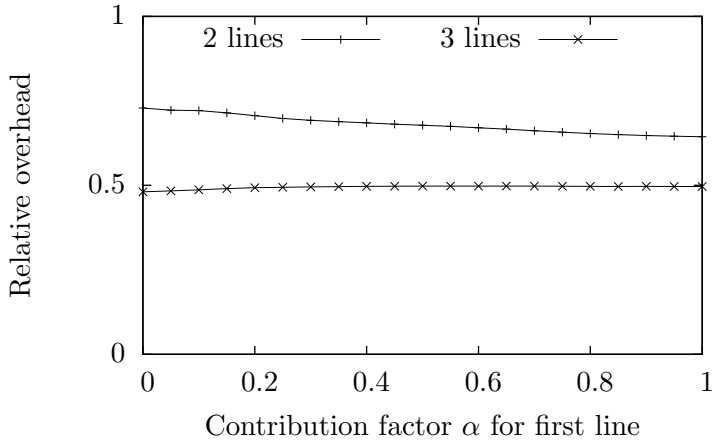



Figure 4.4: Relative overhead given $\alpha$ factor on first line

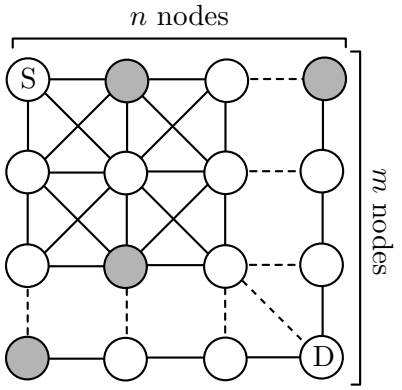



Figure 4.5: Grid topology with $n \times m$ nodes

ing on the contribution factors, since the traffic is merely just moved in the network (Fig. 4.6). The overhead first increases with the fraction of nodes with $\alpha = 0.1$ then decreases until it reaches for every node with $\alpha = 0.1$ the same value as every node with $\alpha = 1$.

This is due to the fact that contribution factors are relative. Having every node in the network with $\alpha = 0.1$ produces the same result as having every node with $\alpha = 1$: in each case, contributions of nodes are considered equal. The slight increase is linked to lower path diversity in the network, forcing to send more redundant combinations to achieve the same reliability.

## 4.6 Conclusion

To handle the issue of the cost of coding, we proposed an extension of the redundancy algorithm algorithm presented in chapter 3.

The algorithm still works in accordance with link loss and application-tolerated loss, but it also takes into account the reported capacities of each node. Redundancy is added to guarantee a maximum applicative loss during decoding at destination, while trying to move the coding operations in priority to nodes reporting the capacity to handle them.

The evaluation we have conducted shows that the coding operations are moved across the

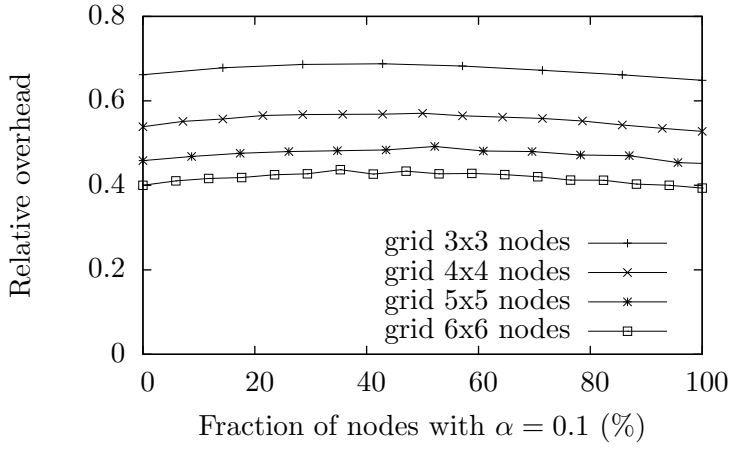



Figure 4.6: Relative overhead given fraction of nodes with $\alpha = 0.1$

network as expected while the overhead is kept low.

# Part II

# Network coding and TCP

# Chapter 5

# Fairness and interaction between network coding and TCP

## 5.1 Objectives

The Transmission Control Protocol (TCP) performs poorly over lossy links because most implementations interpret packet losses as congestion signals [2]. Wireless links can be subject to obstacles or interferences that result in packet losses, leading to reduced TCP performances.

New approaches using network coding have emerged to deal with losses in wireless networks, like Pipeline Coding [10] or TCP/NC [11]. As a form of erasure code, they aim at masking link losses from TCP, in particular by adding redundancy. Thus, they allow to improve its performance.

In parallel, inter-flow coding can combine data and acknowledgements. When a TCP flow transmits data in one direction, acknowledgements flow in the other direction. In a wireless half-duplex network, *e.g.* like WiFi, this behaviour is inefficient since acknowledgements reduce network capacity available for the data flow.

Inter-flow network coding can be used to opportunistically combine TCP segments with TCP segments from the same flow but in the reverse direction, such in Combo Coding [21]. This reduces the auto-interference phenomenon and allows to increase the capacity available to the data flow, hence increasing its throughput.

As adding a network coding layer tends, by design, to conceal network characteristics from TCP, unwanted side effects can appear. TCP uses the well-known AIMD congestion control algorithm which, in most implementations, uses packet losses as a congestion signal. However, intra-flow network coding, like pipeline coding [10] [11], is a good way to mask link losses but potentially also congestion losses from the upper layer, and thus making TCP flows over network coding greedier.

## 5.2 TCP and fairness considerations

Since TCP is by far the most widely used transport protocol on the Internet, several works have naturally focused on its interation with network coding.

TCP/NC [11] introduces a layer between IP and TCP that implements network coding over IP and tricks TCP mechanisms to produce desired results. The source sends linear combinations of all packets in the congestion window, in a pipeline coding way, and the receiver does not actually acknowledge decoded packets but only degrees of freedom in the form of *seen* packets.

TCP/NC translate random losses as longer RTT, therefore the losses are masked and the lossy behaviour of the link appears to both ends as a virtual queueing delay, thus interacting well with TCP Vegas. It has been demonstrated TCP/NC is able to react properly to congestion because congestion losses are not independant but correlated, as shown in its model [32]. The solution is technically elegant, however, it breaks compatibility with plain TCP and suffers from middleboxes traversal issues. In particular, it is unable to go through middleboxes that interpret TCP flows, such as transparent proxies, WAN optimizers or firewalls performing Deep Packet Inspection, and this may prevent its widespread deployment.

These protocols have been proven fair when two coded flows of the same type are competing, but the possible fairness issue when competing with non-coded flows, caused by coded flows being less sensitive to losses, has been more or less left aside.

The underlying issue here is the complex problem of distinguishing link losses and congestion losses. Different approaches have been attempted to solve the problem in the case of non-coded flows. An estimation of RTT can be directly used like in Non-Congestion Packet Loss Detection (NCPLD) [33], with a threshold to assume congestion loss. TCP NewReno-LP [34] and Veno [35] and TCP Vegas's mechanism to estimate queue size, and assume congestion when queue size is high enough. Packet jitter also carries information about congestion state. Jitter-based TCP (JTCP) [36] computes the average of the inter arrival jitter during one RTT, which is proportional to the queueing speed. When 3 duplicate ACKs are received, if the estimated queueing speed is less than the sending speed, then congestion is assumed.

For coded flows, an enhancement of TCP/NC [37] has been proposed to weaken the redundancy when congestion is probable using a loss differentiation scheme based on the Vegas algorithm. CTCP [38] takes a different approach by introducing a whole new coded transport protocol, with its own new coding-aware block-based congestion control based on RTT: the higher the RTT, the more the coding window is decreased when a loss occur. This radical approach comes at the cost of a difficult deployment, as it aims at replacing TCP.

## 5.3 Sensitivity to losses and unfairness

When running TCP over Pipeline coding, the coding layer acts as a transparent Forward Error Correction (FEC) to TCP 6.4, effectively hiding losses to the TCP layer.

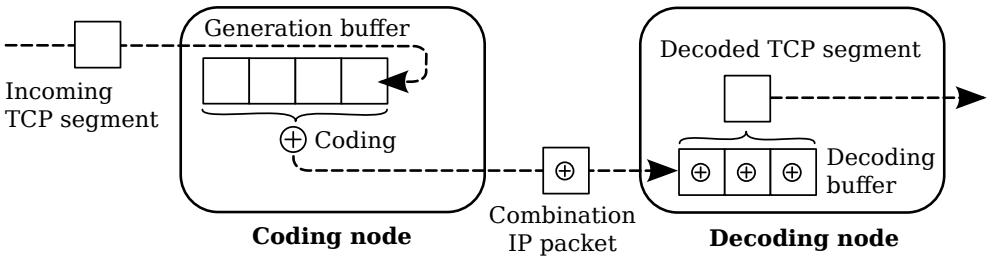



Figure 5.1: TCP over pipeline coding

### 5.3.1 Link and congestion losses

Since losses are indistinctively hidden from TCP by pipeline coding, chances are some part of the congestion losses do not actually trigger TCP window reducing mechanism. This situation get worse when the code redundancy factor increases. Verifying this claim is pretty simple. After setting up a simple bottleneck topolgy with 4 nodes and no link losses (Fig. 5.2) in the *ns-3* network simulator, we monitor packet drops on queue overflow, which correspond to congestion losses.

Then, we compare the window evolution between TCP NewReno without coding and TCP NewReno over coding in an extreme situation, with generation size $g = 64$ and redundancy ratio $r = 1.25$. In our implementation, all intermediary nodes drop redundant packets and recode innovative ones with redundancy $r$, so on each link the actual code redundancy is $r$, independently of the loss rate on preceding links.

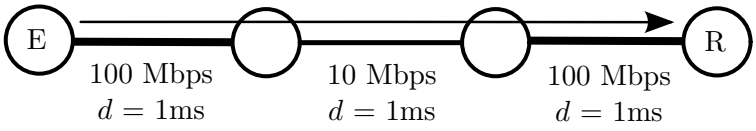



Figure 5.2: Simulated bottleneck topology

On Fig. 5.3 are represented the evolutions of congestion windows, and vertical bars mark the times when a packet is dropped because of queue overflow, *i.e.* when a congestion loss occurs. The observable behaviour of TCP congestion control mechanisms is radically different between the coded case and the non-coded case.

Whereas TCP without coding only need one congestion loss to detect the congestion (by receiving duplicated ACKs) and to react by dividing its window, it needs dozens of them to react over coding, as losses are masked from TCP, therefore congestion detection needs an entire coding generation to be lost, and it can happen only when more than $(r-1)g = 16$ packets are lost. For this reason, overcoded flows are slower to react to congestion than non-coded ones.

One issue is that some overcoding is required, as redundancy should not be set as low as $r_{\min} = \frac{1}{1-p}$ (where $p$ is the average measured loss rate) in order to compensate the losses on average, some more redundancy is necessary to avoid losing generations that statistically encounter more losses or to accommodate a slight decrease in link quality. For instance, ComboCoding's authors use in their redundancy adaptation algorithm a ratio $r = 1.4 + \frac{1}{1-p}$ [21]. A higher redundancy value can lead to coded TCP not reacting well to congestion.

### 5.3.2 Case of competing flows

Let's consider two flows competing for the bandwidth on a network. The first one is TCP running over pipeline coding, and the second one is a not coded TCP flow. Two different factors hinder the non-coded flow and prevent it from using its whole share of available throughput:

- Link losses are hidden from the coded flow but not from the non-coded one. Threrefore, the congestion control of the second flow interprets them as congestion losses : random losses on a link also result in duplicated ACKs or timeouts, as a result the TCP sender adjusts its window size as if a congestion was present on the network. TCP over coding does not suffer from those losses, so it keeps a larger part of the available throughput.

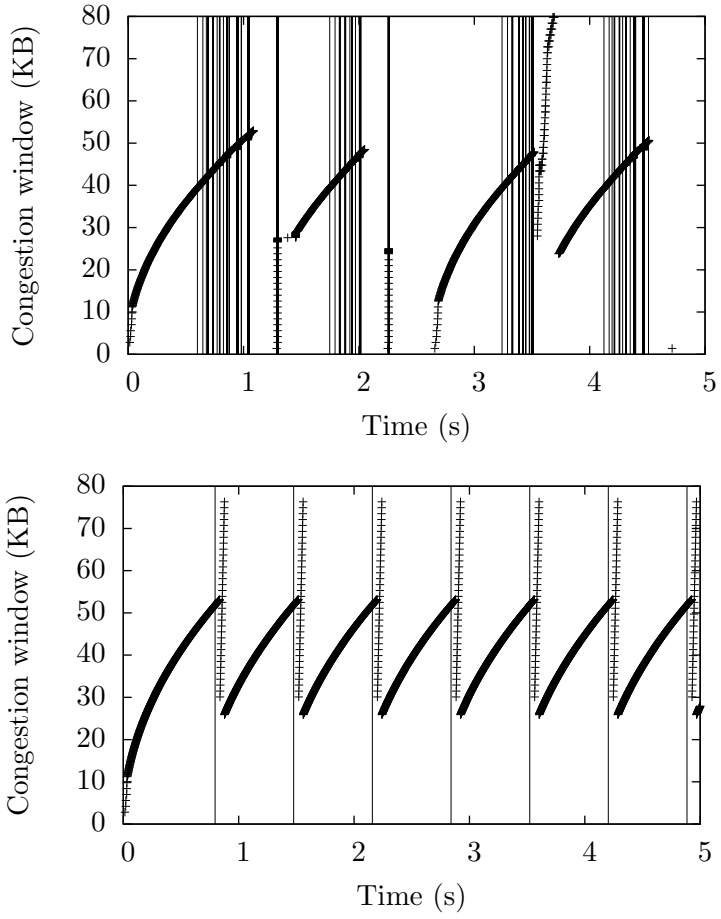



Figure 5.3: Congestion window with (top) and without (bottom) coding

- Some congestion losses are hidden from the coded flow, but none of them are hidden from the non-coded one, so the second flow is more sensitive to actual congestion in the network and reacts faster, leading to a larger part of the throughput for the first flow.

The first mechanism is desired, as it is a simple result of TCP working better over coding, but the second one is clearly undesired. We want to measure which part from the better performance of a coded flow comes from its lower sensibility to link loss and which part comes from higher sensibility of concurrent flows to congestion.

We simulate a simple cross topology (Fig. 5.4) with the simulator *ns-3*.

With a uniform loss probability $p = 1\%$, we run data transfer across two TCP flows, one with coding and one without. Note in this section, the loss probability $p$ is not expressed at a physical level, but after retransmissions on the MAC layer, so $p$ is the actual average loss rate experienced by TCP without coding.

The average throughput over 10 minutes is recorded for each of the two flows in situations with a different coding ratio $r$ for pipeline coding. The generation size is constant $n = 16$. Note we measure the throughput at application level, so measured values represent in reality the goodput.

Results on Fig. 5.5 show the performance with coding is actually worse than without, when

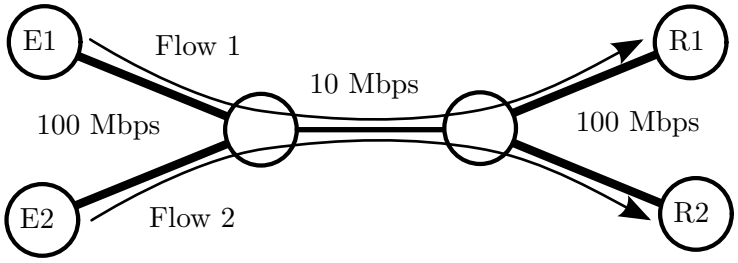



Figure 5.4: Simulated bottleneck topology. All links have 1 ms delay and uniform loss with probabilty $p$.

$r$ is very low this can be explained simply : when the flow has not enough redundancy it tends to lose entire generations, causing TCP to timeout rather than detecting a loss by receiving duplicated ACKs. The difference increase between flows as $r$ increases, the coded flow taking gradually the largest part of the available bandwidth.

However, there seems to be a limit and the coded flow does not take it all, even with relatively high coding ratios like $r = 1.5$. It means the correlation of congestion is sufficent to make the coded flow react at some point.

Yet, the behaviour does not depend on generation size, as shown on Fig. 5.6. Whereas generation size is a key parameter, since it should be set higher enough to not be sensitive to statistically localised losses but low enough to reduce then computational overhead, it is not relevant to flow fairness.

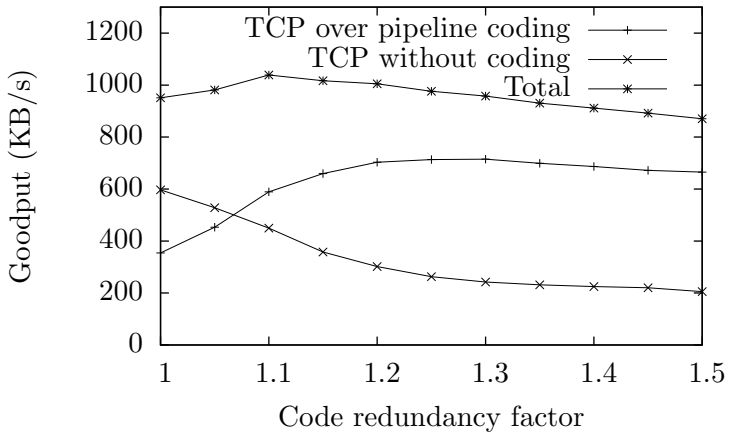



Figure 5.5: Goodput comparison between concurrent flows, one with coding at different coding redundancy factors and one without ($p = 1\%$)

Running the two flows over different loss rates (Fig. 5.7) shows without surprise the throughput for the non-coded flow decreases when losses increase whereas the one of the coded flow reach a maximum as the other flow's throughput decreases. We can see that at a low loss rate, the coded flow leaves some room to the other one. However, it is not clear if part of its throughput is taken at the expense of the non-coded one.

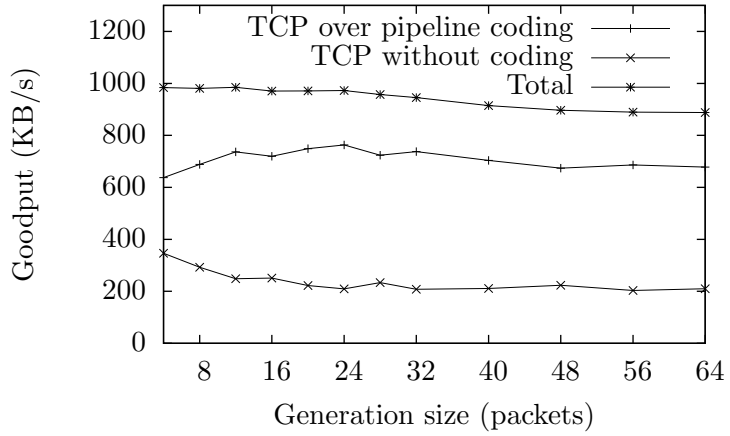



Figure 5.6: Goodput comparison between concurrent flows, one with coding at different generation sizes and one without ($r = 1.25$, $p = 1\%$)

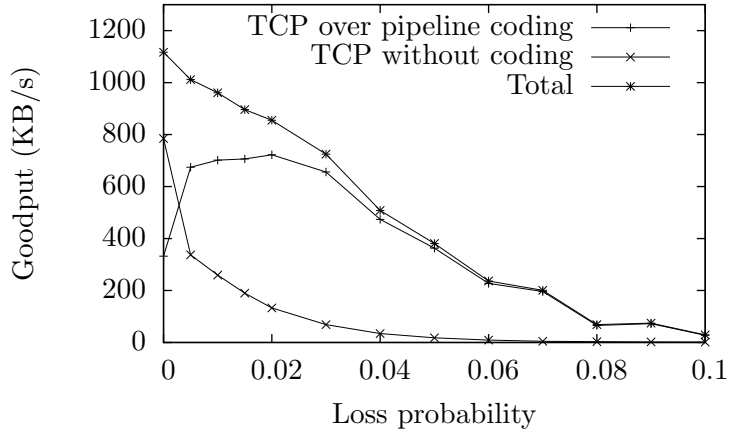



Figure 5.7: Goodput comparison between concurrent flows at different loss rates ($g = 16$, $r = 1.25$)

## 5.4 Measuring fairness

Our goal is to obtain a measure that is sensitive to the throughput that coded flows unfairly take at the expense of not coded ones, but not sensitive to coded flows performing better without impacting not coded ones.

Figure 5.8 shows graphically in a simple case with two flows the part of throughput we consider as unfairly taken : in the second case, with two non-coded flows, the whole capacity is not used because links are too lossy, whereas in the first case, the coded flow may get more bandwidth by using the whole capacity, but what is taken from the first non-coded flow is considered unfair.

To solve this issue, we introduce a simple modified fairness index taking into account the better performance of coded flows over non-coded ones.

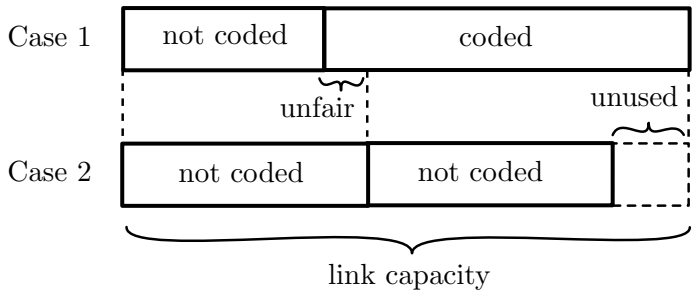



Figure 5.8: Comparison of capacity utilization on a lossy link between a first case with a non-coded and a coded flow, and a second case with two non-coded flows

### 5.4.1 Jain's fairness index

Let's consider $n$ flows, $x_i$ being the throughput of the $i$th flow. Jain's fairness index $J$ [39] rates the fairness of this allocation with a value between $\frac{1}{n}$ (worst case) and 1 (best case, all users receive the same allocation).

$$J(x_1, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$$

The best case for $J$ corresponds to a uniform allocation, but a uniform allocation is not neccessarily optimal with coded and non-coded flows. The index reflects the actual throughput difference and does not take into account the potentially better performance of coded flows. Therefore, we need to define an ideal *fair allocation* taking into account that coded flows can perform better without impacting non-coded ones.

### 5.4.2 Fair allocation with coded and non-coded flows

For a given set of flows, we define a *fair allocation* as an allocation where :

- The allocation for the subset of coded flows is fair according to Jain's index.
- The allocation for the subset of non-coded flows is fair according to Jain's index.
- No non-coded flow can get less throughput than it would get if coded flows were replaced with non-coded ones.

Intuitively, a fair allocation as defined previously is not necessarily fair according to Jain's index, as a coded flow can get more throughput than a non-coded one, but in a fair allocation, no coded flow can get more throughput at the expense of non-coded ones.

### 5.4.3 Formalization

Let $A$ be an allocation with $n$ flows competing on a lossy path, the first $k$ ones are coded and the $n - k$ others are not coded ($k > 0$). $x_i$ is the throughput the flow $i$ get in the allocation $A$.

Let $A'$ be an allocation with the same characteristics but where all coded flows are replaced with non-coded ones. $x'_i$ is the throughput the flow $i$ get in the allocation $A'$ (Fig. 5.4.3).

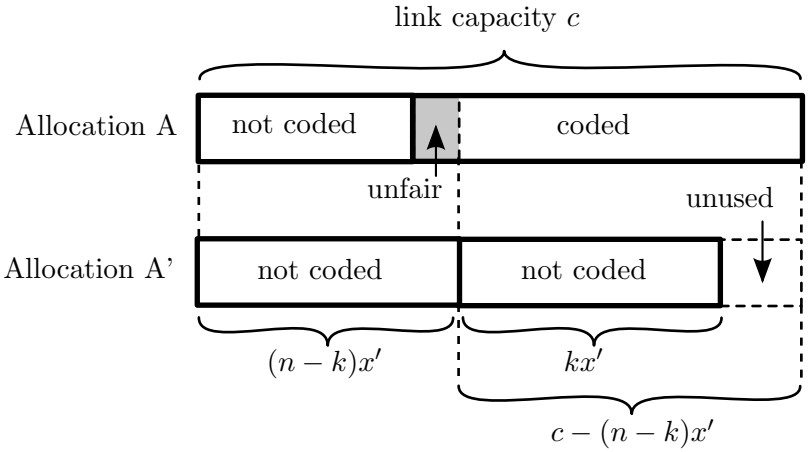



Figure 5.9: Formalization of capacity utilization on a lossy link between allocation $A$ and allocation $A'$ where coded flow are replaced with non-coded ones

The capacity $c$ is used in the allocation $A$. We assume here that the presence of at least one coded flow means more bandwidth is used compared to the allocation $A'$, as coded flows should perform better on lossy links.

$$c = \sum_{i=1}^{n} x_i \geqslant \sum_{i=1}^{n} x'_i$$

Let $y_i$ be the throughput the flow $i$ should get in a fair allocation according to 5.4.2.

A non-coded flow can get less throughput than a coded flow, but it remains fair if every non-coded one gets the same absolute throughput as it would if coded flows were replaced with non-coded flows. It means $\forall i \in \{k+1, ..., n\}\ y_i = x'_i$.

Because of the fairness of TCP congestion control, as we consider identical network charasteristics for every flow (RTT, losses...), we get :

$$J(x'_1, ..., x'_n) = 1$$

$$\forall i \in \{1, ..., n\}\ x'_i = x'$$

So the share of every non-coded flow in the fair allocation should be:

$$\forall i \in \{k+1, ..., n\}\ y_i = x'_i = x'$$

Coded flows should get the same share of the fair allocation :

$$\forall i \in \{1, ..., k\}\ y_i = y$$

The whole available capacity should be used :

$$\sum_{i=1}^{n} y_i = ky + (n-k)x' = c$$

So the share of every coded flow in the fair allocation should be:

$$\forall i \in \{1, ..., k\}\ y_i = y = \frac{1}{k}(c - (n-k)x')$$

In particular if $n = 2$ and $k = 1$, it simply means the coded flow should use the whole bandwith that would remain if it was not coded:

$$y_1 = c - x'$$

$$y_2 = x'$$

### 5.4.4 Modified fairness index

We have seen in the previous paragraph that, in a fair allocation, a coded flow should get $\frac{1}{k}(c - (n-k)x')$ when a non-coded flow should get $x'$, $x'$ being the average throughput a flow would get in the same situation if coded flows where not coded.

Let's define $\lambda$ the ratio between the throughput of a non-coded flow and the one of a coded flow in a fair allocation :

$$\lambda = \frac{kx'}{c - (n-k)x'}$$

If we multiply the throughputs $y_1, ..., y_k$ of the coded flows by a factor $\lambda$, Jain's index for the fair allocation becomes 1 :

$$J(\lambda y_1, ..., \lambda y_k, y_{k+1}, ..., y_n) = 1$$

So we introduce a modified fairness index $J'$ taking this correction into account, *i.e.* the throughputs $x_1, ..., x_k$ are multiplied by $\lambda$ :

$$J'(x_1, ..., x_n) = \frac{(\lambda \sum_{i=1}^{k} x_i + \sum_{i=k+1}^{n} x_i)^2}{n(\lambda^2 \sum_{i=1}^{k} x_i^2 + \sum_{i=k+1}^{n} x_i^2)}$$

This modified fairness index has the same properties as Jain's fairness index, but the best case corresponds to a fair allocation as defined in 5.4.2 and not a uniform allocation. It rates the fairness with a value between $\frac{1}{n}$ (worst case) and 1 (best case, the allocation is fair).

In particular if $n = 2$ and $k = 1$:

$$\lambda = \frac{x'}{c - x'}$$

$$J'(x_1, x_2) = \frac{(\lambda x_1 + x_2)^2}{2(\lambda x_1)^2 + 2x_2^2}$$

For instance, let's consider a coded flow competing with a non-coded flow on a lossy link. The measured throughputs are $x_1 = 7$ Mbps for the coded flow and $x_2 = 3$ Mbps for the non-coded flow, so $c = 10$ Mbps. A similar experiment but with 2 identical non-coded flows give us $x'_1 = 4$ Mbps and $x'_2 = 4$ Mbps, so $x' = 4$ Mbps and $\lambda = \frac{2}{3}$. We eventually get $J'(x_1, x_2) \simeq 0.95$ (nearly fair) whereas $J(x_1, x_2) \simeq 0.86$.

Applying this new index to the previously measured throughputs gives us a more precise idea of the actual flow fairness. Note computing this index also requires running non-coded TCP flows in the same configuration, as their average throughput is required to compute $\lambda$.

Figure 5.10 highlights the allocation is actually fairer in reality than what Jain's index shows, as the coded flow takes a part of the resource the non-coded flow could not take anyway. As a side effect, the maximum fairness is displaced to higher redundancy (here roughly from 1.05 to 1.1). In the case of higher loss rate (Fig. 5.11), the two indices give rather different results. This is caused by Jain's index not taking into account the very poor performance of TCP at important loss rates, whereas the modified index does. We can see that at some point, when losses are too important, the non-coded flow performs so badly that it is actually fair for the coded flow to take an overwhelming part of the throughput.

When the redundancy factor $r$ approaches higher values, fairness does not drop and stays relatively high around 0.85. This is a good sign, as it indicates even with too much redundancy, coded TCP flows do not starve non-coded ones. It means congestion losses are correlated enough to trigger a generation loss and TCP congestion control algorithm.

It is interesting to note that the maximum fairness is achieved when $R = 1.1$, and it corresponds to the session giving maximum cumulated throughput. This is logical in the sense that fairness is achieved when the two flows give simultaneously the best possible performance. A maximum fairness is achieved for $R = 1.25$ near $p = 0.02$ for similar reasons.

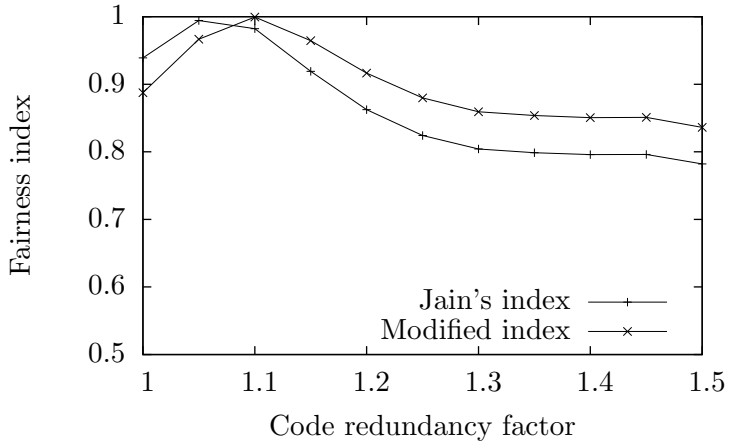



Figure 5.10: Jain's index at different coding factors for a coded and a non-coded flow ($p = 1\%$)

## 5.5 Conclusion

In this chapter, we highlighted the fairness issue raised by the implementation of intra-flow network coding with TCP flows. The issue arises because concealing link losses interferes with TCP congestion control, since it also conceals congestion losses.

To evaluate the impact of pipeline coding on fairness, we indroduced a simple specific index taking into account the better performance of coded flows on lossy links.

Our results show that unfairness exists but its impact is relatively limited, because even with high redundancy factors a coded flow does not starve a non-coded one, indicating that

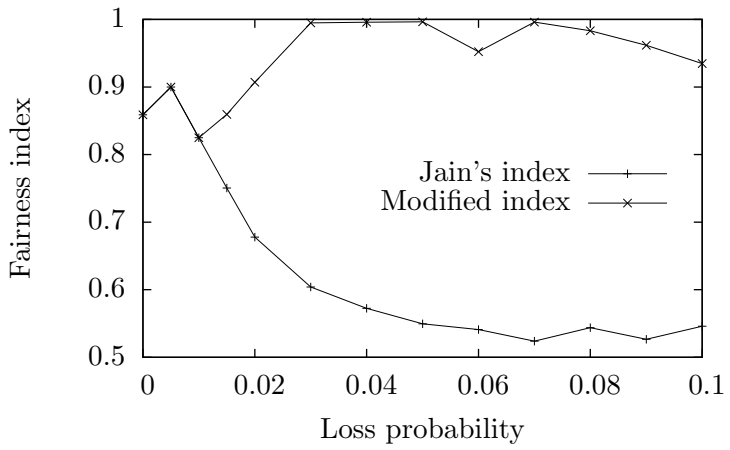



Figure 5.11: Jain's index at different loss rates for a coded ($g = 16$, $R = 1.25$) and a non-coded flow

congestion losses are correlated enough to cause the coding to fail and TCP to react to the congestion.

# Chapter 6

# Increasing reliability of MPTCP over network coding

## 6.1 Objectives

Multi-path TCP (MPTCP) is a recent TCP extension standardized in RFC 6824 [40] by the IETF. It aims at spreading a single TCP connection over several paths with different addresses, yet it hides the involved complexity from the upper layer by keeping the standard TCP socket API. It aims at optimizing network resource usage and providing better throughput and robustness.

MPTCP was at first designed to provide performance benefits when deployed in datacenter environments. Typically, MPTCP targets multihomed hosts in data centers for the purpose of increasing inter-datacenter bandwidth.

Recently, its interest has been demonstrated for mobile devices featuring multiple radios. For instance, smartphones connected to both WiFi and LTE can take advantage of it to provide stable network performances in spite of intermittent degradations or failures on either channel and to enhance handover when switching between WiFi and LTE. Indeed, it showcases several advantages: it improves connectivity and quality of service, increases throughput by allowing the use of multiple interfaces for data transfer, and seamlessly handles handover and traffic offload from congested radio access networks [3].

From a network point of view, MPTCP runs fully TCP-compatible sub-flows for different pairs of IP addresses, on which the original application data flow is mapped. This complete retro-compatibility allows MPTCP to be deployed without middlebox traversal issues.

In the same way that TCP is sensitive to link loss, MPTCP suffers from performance degradation when used over paths experiencing random losses. Not only lost packets must be retransmitted, but the congestion control algorithm tends to misinterpret link losses as congestion signals [41], leading to a uselessly reduced congestion window and a lower throughput.

In this section, we study the impact of pipeline coding on MPTCP performance. We implemented an experiment using the MPTCP reference implementation [42] and developed a network coding layer at IP level. We believe that using network coding at IP level is a relevant approach as it provides TCP retro-compatibility and avoids middlebox traversal issues. The network coding layer allows to compensate for link losses preventing both congestion windows

reduction and head-of-line blocking issues at the scheduler. We conducted a performance evaluation of coded MPTCP on a realistic emulated network. A comparison of MPTCP performance with and without coding is provided.

## 6.2 Multi-path TCP over network coding

TCP performs poorly over lossy links because it interprets packet losses as congestion signals [2]. As MPTCP relies on the same mechanisms. it suffers from the same issue.

Intra-flow network coding is an interesting solution for reliable transfer over lossy networks. The idea behind intra-flow network coding is to send random linear combinations of outgoing packets. To compensate losses, more combinations than original packets are generated. The redundancy factor *i.e.*, the enforced ratio between sent combinations and original packets, is a critical parameter. It should be large enough to compensate link losses and guarantee enough combinations at reception to ensure the packets are decoded. However, setting it too high can lead to useless overhead and network congestion.

Therefore, the idea is to have MPTCP subflows running over pipeline coding, as shown on figure 6.1.

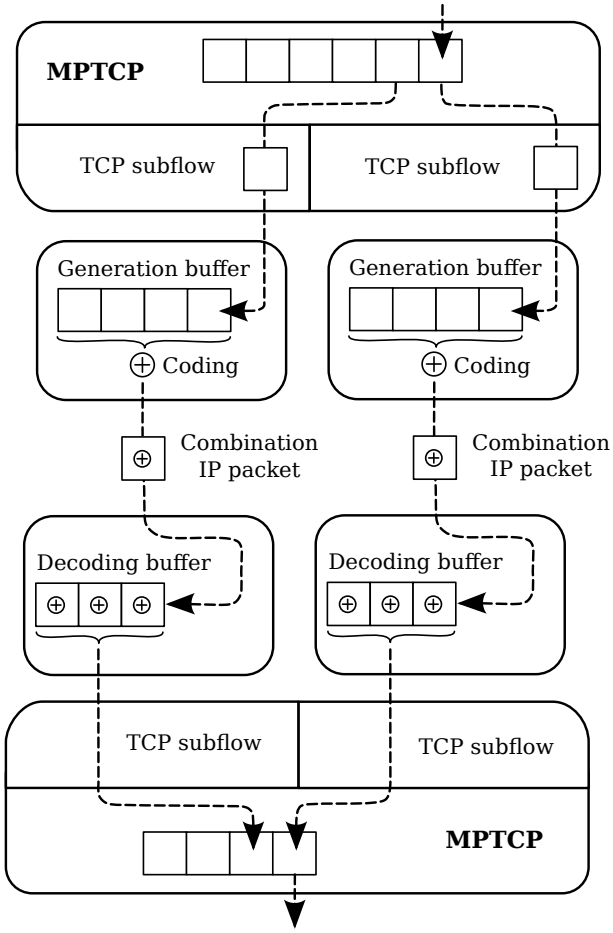



Figure 6.1: MPTCP with two subflows running over intra-flow pipeline coding

The principle is to keep the MPTCP stack untampered and to run every subflow on a pipeline

coding session. Packets from each subflow are put in a generation buffer and new combinations are sent on every incoming packet, using the progressive coding of pipeline coding. When the generation buffer reaches the generation size, the buffer is cleared to start a new generation.

On the receiver side, packets from each subflow are decoded independently. When a packet can be decoded, it is immediately passed to MPTCP.

The redundancy ratio, *i.e.*, the number of combinations for every original packet, is adjustable, which will allow to observe its impact on performance.

Network coding should have a positive impact on MPTCP when running over lossy links, because it not only protects the TCP subflows from link losses, but it also prevents scheduling issues when react to losses on subflows by retransmitting on another link.

## 6.3 Emulated network setup

We conducted our tests on the MPTCP reference implementation for the Linux kernel developed by Université Catholique de Louvain, version 0.90. In our setup, this real-world implementation runs over emulated links *i.e.* in realtime. Network emulation is achieved with ns-3 simulator running in realtime mode. Two User-Mode Linux (UML) virtual machines, one for the client and one for the server, are installed on the host alongside the simulator. Traffic from the interfaces of the two virtual machines is routed to the corresponding nodes in ns-3. When a file transfer is run between the machines, subflows are opened across the emulated network (figure 6.2).

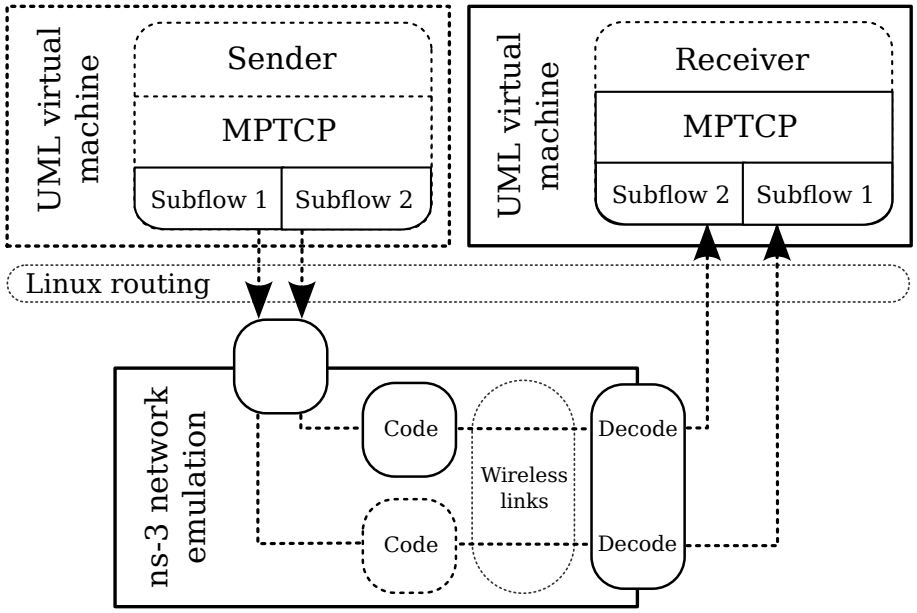



Figure 6.2: Network emulation setup with ns-3 and two User-Mode Linux virtual machines

The emulated network consists of four nodes: two nodes as wireless access points, one for the server and one for the client with two interfaces. The client and server nodes use ns-3 tap bridge feature to make their interfaces available from the Linux host, and traffic is routed to and from the corresponding virtual machines with specific routing tables on the host. Independent wireless links are emulated between the client node and the access points (figure 6.3).

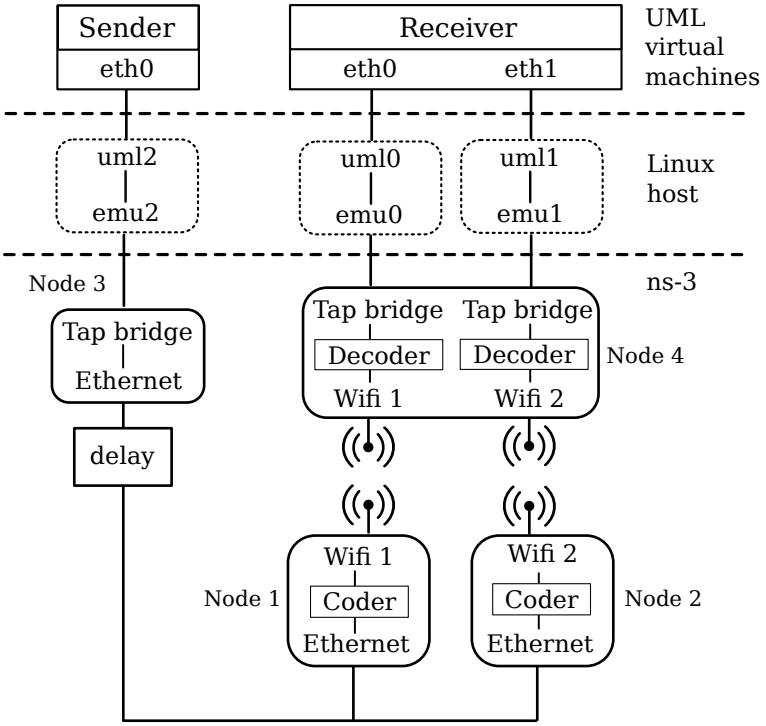



Figure 6.3: ns-3 emulation internal details

Wireless access points implement pipeline coding, and the client independently implements decoding on both its wireless interfaces. Pipeline coding is implemented at IP level with specific coding and decoding buffers 6.4.

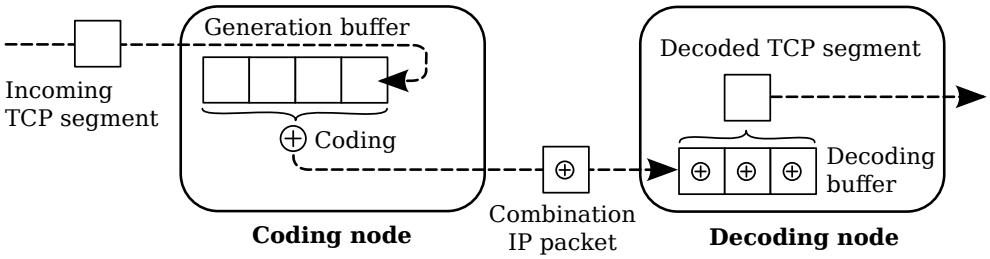



Figure 6.4: Pipeline coding as implemented in ns-3 nodes

## 6.4 Performance evaluation

### 6.4.1 Influence of loss rate and redundancy

Figure 6.5 shows the application goodput for an MPTCP flow running on two links with and without coding for different loss rates and with RTT 100ms. Flows are run for 60 seconds for each link quality and values are averaged over 10 experiments. As loss rate increases, the flow without coding collapses whereas the one with network coding is protected, offering constant goodput until high loss rates.

We also note that due to the redundant coded packets, the coding shows an overhead in

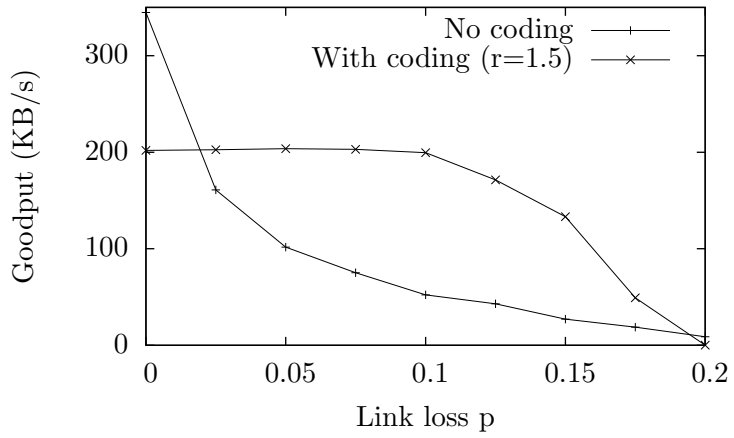



Figure 6.5: Application goodput for MPTCP on two links given loss rate $p$ with 100ms RTT (generation size 8)

terms of capacity when link loss rate is near zero. Redundancy is a critical parameter for intra-flow network coding, it represents a serious tradeoff in terms of performance. Higher the redundancy, higher the overhead, but better the protection against random losses.

Figure 6.6 compares the application goodput for different redundancy factors $r$. Increasing redundancy makes the maximum loss rate before the goodput collapses increase. As redundancy increases, the overhead increases and less capacity is available for the application. The goodput at lower link loss rates is lower with higher redundancy factors.

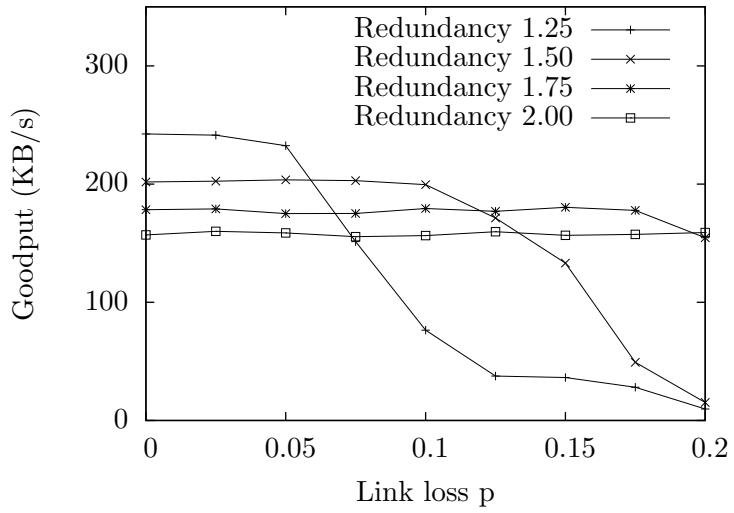



Figure 6.6: Application goodput given loss rate comparison given redundancy factor $r$ (generation size 16)

However, in terms of goodput, it is better to have "over-redundancy" than "under-redundancy". Figure 6.7 shows the evolution of goodput given the chosen redundancy factor at constant loss rate $p = 0.1$. When the redundancy is too low, not enough losses are corrected and MPTCP congestion control tends to be triggered uselessly, and this proves very harmful to the flow. Setting it too high simply creates more overhead without a notable benefit on congestion control, creat-

ing overhead and reducing the useful flow capacity. However, we notice that "over-redundancy" does not have an impact as critical as "under-redundancy".

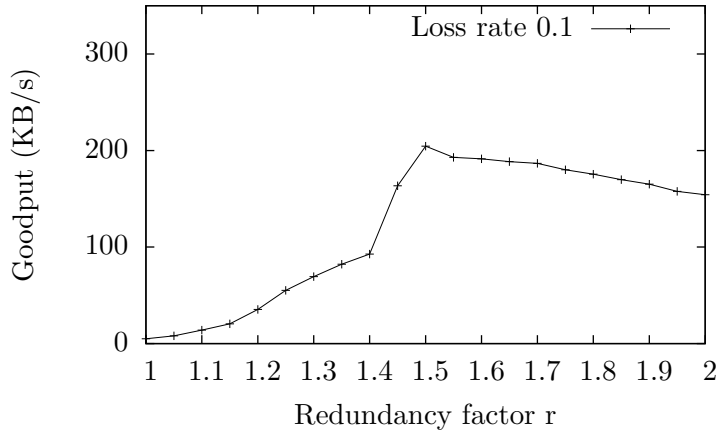



Figure 6.7: Application goodput given redundancy factors $r$ for loss rate $p = 0.1$ (generation size 8)

### 6.4.2 Influence of generation size

Apart from code redundancy, code generation size also has an impact on performance. Figure 6.8 measures goodput using generation sizes 8 and 16. High generation sizes tend to cause more overhead because of longer decoding delays, but they have the benefit of performing more reliably when loss rate is just sufficient to compensate link loss, because they are less sensitive to local loss rate variations. Overall, small generations seem more adapted to MPTCP.

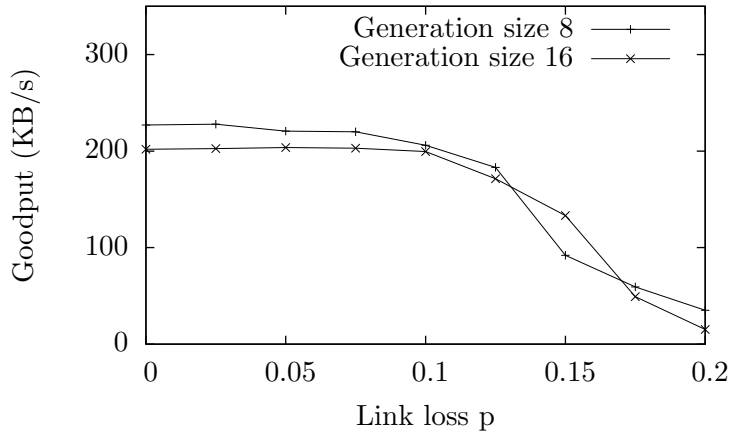



Figure 6.8: Application goodput for MPTCP given loss rate $p$ for different generation sizes (RTT = 100ms, $r = 1.50$)

### 6.4.3 Influence of Round-Trip Time

The benefit of coding is even more visible as RTT raises, since TCP takes more time to recover from losses. Without coding, higher RTTs worsen the loss issue and the goodput collapses even faster. The flows using network coding show a performance virtually independent of RTT at low losses, even if it is sligthly affected at higher loss rates (figure 6.9 and figure 6.10).

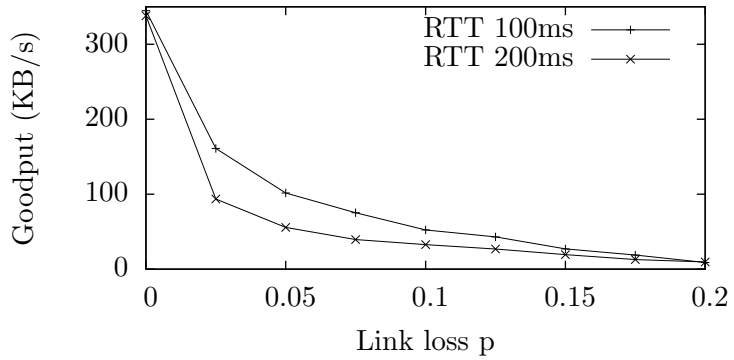



Figure 6.9: Application goodput for MPTCP without coding given loss rate $p$ according to RTT

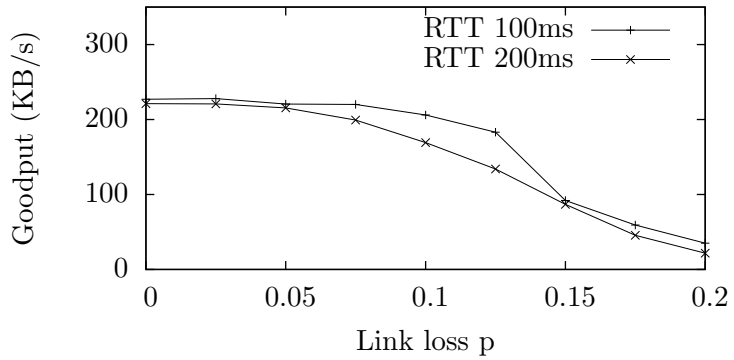



Figure 6.10: Application goodput for MPTCP with coding given loss rate $p$ according to RTT ($r = 1.5$)

## 6.5 Conclusion

In this chapter, we proposed an enhanced MPTCP solution using network coding. Our goal is to increase MPTCP reliability without tampering with the MPTCP stack, maintaining TCP retro-compatibility and avoiding middlebox traversal issues.

The tests carried out on our network emulation setup show that the MPTCP goodput is significantly improved when running on top of pipeline coding. Besides, the results show the impact of the redundancy factor on performance. Redundancy is indeed a tradeoff between reliability and overhead, and a correct value should be chosen to get optimal performance, even if the tests show that over-redundancy is less harmful than under-estimated redundancy.

This works allowed to create an emulation platform that will be useful for our research

team to carry out tests on MPTCP. In particular, the platform will allow extensive tests with MPTCP running in more realistic conditions.

# Chapter 7

# Integration of network coding in the MPTCP protocol

## 7.1 Objectives

In this chapter, we investigate network coding as a potential solution to the head-of-line blocking issue. Indeed, MPTCP uses a global window that is mapped to the windows of the subflows, and a common issue encountered with this approach, especially with unreliable wireless links, is head-of-line blocking.

It happens when a link suddenly flickers or experiences delays: the global MPTCP window cannot move forward because packets scheduled on the failing link are missing (Fig. 7.1). Until those packets eventually get through or are reinjected on working links, the connection is locked even if other TCP subflow links show good performance. For instance, this scenario can happen when a user is downloading a video stream from both WiFi and LTE moves away from the WiFi acces point. The degradation of the WiFi link quality due to the increased distance, leading to a flickering link, could delay the entire MPTCP data flow and freeze the video.

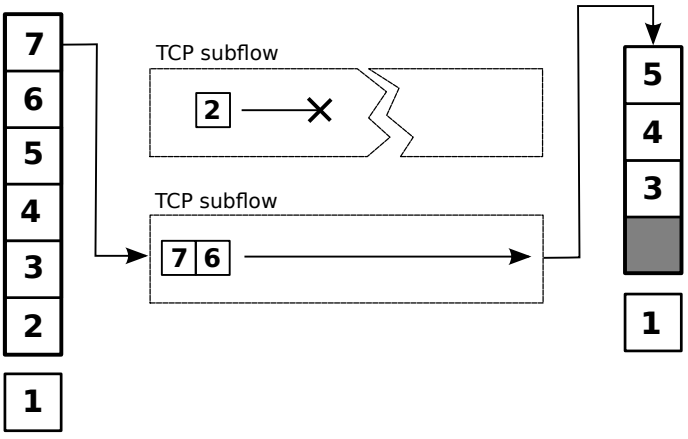



Figure 7.1: Head-of-line blocking issue encountered with MPTCP

Our goal is to increase the resilience and improve the overall performance of MPTCP by solving the head-of-line blocking issue.

To this end, we define and test a practical MPTCP extension, MultiPath Coded TCP (MPC-TCP), that enables the TCP source to transmit linear combinations of packets rather than data packets over TCP subflows. On the receiver side, data can be decoded as soon as enough combinations are available, no matter the subflows that were used to transmit them (Fig. 7.2). This scheme offers three advantages: MPTCP retrocompatibility, full TCP compatibility for middleboxes and NAT traversal, and implementation simplicity.

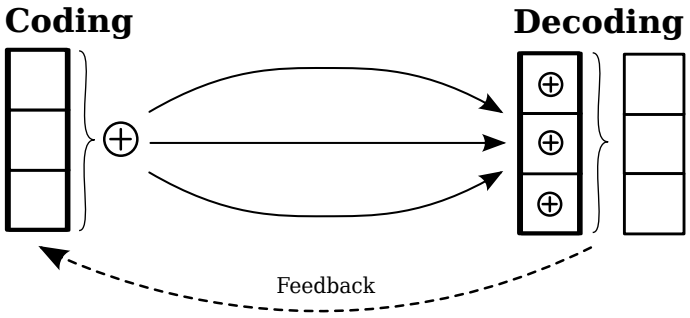



Figure 7.2: Simplified principle of coding over MPTCP

We developped our own implementation in the Linux kernel based on the MPTCP reference implementation v0.90 from Université Catholique de Louvain [42].

Evaluations have been carried out with an emulated network in ns-3 to measuse the capacity of network coding to solve the blocking issue. In this chapter, we only focus on solving head-of-line blocking issues, and kept fully TCP-compatible subflows, preferring retrocompatibility over maximum performance on lossy links.

While in most other implementations, technical constraints are not discussed, practical implementation and compatibility is one of our main concerns.

## 7.2 Network coding integration in MPTCP

TCP performs poorly over lossy links because it interprets packet losses as congestion signals [2]. As MPTCP relies on the same mechanisms. it suffers from the same issue. Not only lost packets must be retransmitted, but the congestion control algorithm [41] tends to misinterpret lost packets due to link loss as a congestion signal, leading to a uselessly reduced congestion window and a lower throughput. Network coding can solve this problem by adding redundancy to the flow, as a form of forward error correction, improving reliability. Since in our case, combinations are transmitted over reliable TCP flows, we don't address this issue.

Another issue is head-of-line blocking (Fig. 7.3). If packets scheduled on one of the paths are delayed or lost, the entire flow at reception is also delayed. Network coding allows to decorellate the flows, as receiving combinations from any subflow carries new information with high probability and thus allows to continue decoding (Fig. 7.4).

Recently, several works have been developed to integrate network coding in MPTCP flows, in order to benefit from protection against link loss and as a solution to head-of-line blocking issues caused by flow segments scheduling on the different available paths.

In MPTCP/NC [43] a modified TCP stack is proposed to solve both issues. Namely, the subflows, instead of running on standard TCP, run over TCP/NC, hence it can be seen as a multi-path extension of the TCP/NC protocol mentioned previously. The protocol has been

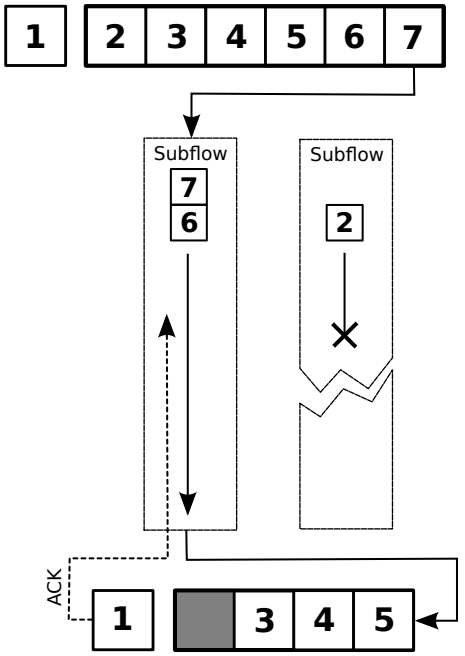



Figure 7.3: Head-of-line blocking issue encountered with MPTCP

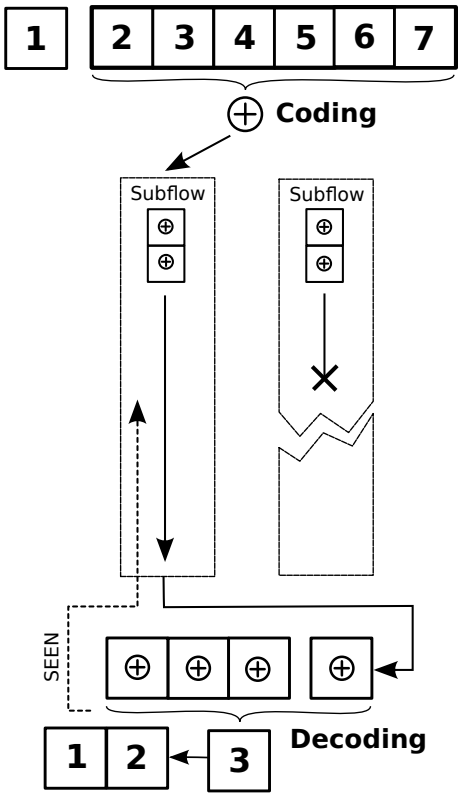



Figure 7.4: Head-of-line blocking solution with network coding

developed to allow better subflow performance in the face of link loss as intra-flow redundancy can be added to compensate for link loss. However, MPTCP/NC presents the same drawback as TCP/NC when it comes to ease of deployment, since the lack of retrocompatibility with TCP

may cause compatibility issues with middleboxes, barring widespread adoption.

In NC-MPTCP [44], coded subflows run in parallel to regular subflows, which is an interesting feature but forces to implement a specifically-designed scheduling algorithm, increasing complexity. The scheme make use of a form of batch, since the decoding process is performed after the decoding matrix has completed to decode the entire generation as a batch, potentially adding unnecessary jitter from the application point of view. The coded subflows are not TCP compatible, which can cause issues with firewalls or middleboxes on restrictive network environments as corporate networks.

Approches trying to use network coding to help head-of-line blocking in multipath transfers exist for other protocols, like SCTP. An inclusive proposition is presented in [45]. The scheme showcases bandwidth, loss and RTT estimators to compute code redundancy and perform data distribution across flows.

The SCTP-CMT protocol in [46] proposes an innovative way to solve receiver buffer blocking for SCTP-based Concurrent Multipath Transfer using network coding and machine learning. Random linear coding is used to decorellate the flows and make the receiver insensitive to packet reordering, whereas a Q-learning algorithm enforces coding redundancy at the sender side to make sure data is decodable at the receiver side.

Even if SCTP provides huge benefits for various kind of applications, it's far for widely deployed, apart as transport protocol for Signaling System #7 (SS7) implementations. For instance, nearly no consumer application make use of it, moslty because it's often poorly handled by middleboxes, especially consumer-grade routers, and because popular operating systems like Microsoft Windows were lacking out-of-the-box support until recently. SCTP-based network coding implementations will undoubtedly inherit this drawback, preventing practical deployment.

## 7.3 Principle

The principle of the proposed protocol is to keep the main architecture of MPTCP and replace the global sending and receiving buffers with random linear coding and decoding windows. The TCP subflows carry combinations rather than data segments but work as normal TCP flows, for compatibility with middleboxes. The MPTCP global acknowledgement mechanism is modified to acknowledge components that shouldn't be part of the coding process anymore (*seen* components) rather than received segments, and the sender uses this information to move its coding window forward (Fig. 7.5).

In MPTCP, the head-of-line blocking issue happens when one of the subflows stops working. In this case, the data mapped to this subflow must be rescheduled at some point, but during the time period, transmission might be blocked because the data is missing at the receiver side and thus the window is prevented from moving forward.

With random linear combinations, the goal is to become independant from the subflows : every received combination adds more information about the data to be transmitted, issues like missing chuncks cannot arise. So, the implementation only has to push combinations in the available subflows without bothering about subflow performance, since every combination that goes through in time should allow the receiver to decode more data. Note global feedback always works because it is piggybacked on the acknowledgements of every subflow.

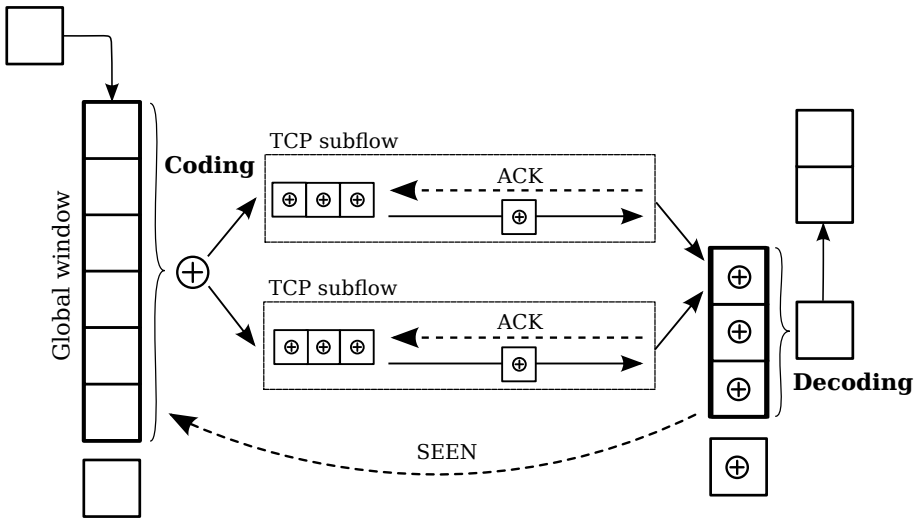



Figure 7.5: Overview of the coding/decoding principle with two subflows

Apart some precise points that will be highlighted in the following, the implementation is exactly the same as MPTCP as defined in RFC 6824 [40]. In particular, no modifications are made to the fields defined for headers and options. Subflows are standard TCP flows, therefore full compatiblity with middleboxes and NATs is guaranteed.

Since we use TCP flows to transmit the combinations, sending redundancy over a flow does not make sense, since flows are reliable and do not experience loss or reordering at coding level. However, we have to deal with TCP delays due to retransmissions.

The proposed protocol MPC-TCP is extensively compatible with MPTCP as defined in [40] and need as little modifications as necessary, since nearly no changes are made to the headers and options. Moreover, the subflows are never coded on a network level and always appear as normal TCP flows from an external point of view. This is the main departure from previous schemes that also attempted to add coding to MPTCP by modifying the TCP subflows. Our approach yields better backward compatibility and easier incremental deployment, particularly with restrictive middleboxes and firewalls. Moreover, the use of MPC-TCP is negotiated during the MPTCP handshake, allowing easy fallback to vanilla MPTCP if necessary. Finally, the use of linear packet combinations makes the complicated mapping and packet bookkeeping system handled by MPTCP unnecessary, so the actual implementation is a lot simplified.

### 7.3.1 Activation

MPTCP uses a special MP_CAPABLE option during TCP handshake to advertise the availability of the protocol. We define a new flag in this option to advertise the availability of MPTCP.

In pratice, the flag C = 0x20, which was affected to cryptographic algorithm negotiation but was unused, is now set to 1 to indicates that coding is available, and after receiving an MP_CAPABLE option with the C flag set to 1, an implementation can use MPC-TCP rather than vanilla MPTCP.

### 7.3.2 Coding

Coding is achieved by sending in subflows linear combinations of segments from the global window rather than mapping portions of it to subflows. The global window $W$ is segmented in chunks of maximum size $\min(\text{MSS}) - 1$, each segment $P_i$ in the window is called a *component* and has an incremental sequence number $i$, called *component number*.

The components are padded with ISO/IEC 7816-4 padding (mandatory first byte is 0x80, optional next bytes are 0x00) to be the same size before coding. This means the combinations are one-byte larger than the largest outgoing segment, so segments are one byte less than the minimum MSS for efficiency, and the MSS advertised to the upper layer must be reduced by one.

When a TCP subflow has sufficient window space available and $W$ is not empty, the sender has the opportunity to push a new combination, and it may generate a new one. However, no more than one combination per component should be sent per subflow at once, because doing so would generate useless redundant combinations.

MPTCP transmits global window mapping information in TCP data packets with an option called Data Sequence Signal (DSS). This option is not modified but the fields `Data ACK` and `Data sequence number` are interpreted differently, since we don't need to map data with MPC-TCP. A new flag C = 0x20 is added using the rightmost bit of the reserved field, set to 1 to indicate the mapped data is a coded combination.

The MPTCP `Data sequence number` field is now used to store the `Combination Identifier` $S_{n,k,N}$. It is now always 64 bits long and contains an identifier describing the vector of coefficients used to compute the linear combination corresponding to the mapped data. The low order 32 bits correspond to the first component number $n$ in the combination, the next 16 bits correspond to the combination length (*i.e.* components count) $|W| = k$ and the highest order 16 bits are a unique nonce $N$.

$$W = (P_n, P_{n+1}, ...P_{n+k-1})$$

$$S_{n,k,N} = \text{identifier}(n, k, N)$$

The nonce $N$ must be repeated as infrequently as possible on a single MPTCP connection. It should be set at an initial random value and incremented by 1 for each generated combination.

$S_{n,k,N}$ can be used to retrieve the first component number $n$ and the combination length $k$. These values alone give the number of each component in the vector, since the components are contiguous. Then, $S_{n,k,N}$ serves as a seed for Donald Knuth's MMIX 64-bits linear congruential generator, whose output non-null 8 most significant bits serve as coefficients. This way, the complete combination coefficients vector can be derivated unambiguously from $S_{n,k,N}$.

$$(c_{n,N}, c_{n+1,N}, ...c_{n+k-1,N}) = \text{coefficients}(S_{n,k,N}, k)$$

Then the linear combination $C_{n,k,N}$ is computed with the coefficients. Each component is treated as a vector over the finite field $GF(2^8)$.

$$C_{n,k,N} = c_{n,N} \otimes P_n \oplus ... \oplus c_{n+k-1,N} \otimes P_{n+k-1}$$

The combination is sent as a MPTCP mapped data chunk with $S_{n,k,N}$ in place of the MPTCP connection-level data sequence. Note the subflows are normal TCP flows with unmodified windows, acknowledgements and retransmissions.

### 7.3.3 Decoding

When $C_{n,k,N}$ is received as a MPTCP mapped data chunk from one of the subflows, $n$ and $k$ are extracted from $S_{n,k,N}$ and $S_{n,k,N}$ is used to derivate the $k$ coefficients.

$$(n, k) = \text{extractnk}(S_{n,k,N})$$

$$(c_{n,N}, c_{n+1,N}, ...c_{n+k-1,N}) = \text{coefficients}(S_{n,k,N}, k)$$

The receiver stores the combination with the previously received ones, and attempts to solve (even partially) the resulting system with $k$ unknowns $(P_n, P_{n+1}, ...P_{n+k-1})$ using Gaussian elimination. If one of the $P_i$ can be expressed as the combination of components with a higher number, i.e. $P_i = \sum_{j>i} P_j$, then the component $P_i$ is *seen*. The number $s$ such as $s-1$ is the highest seen component number is called the *next seen number*. Each time a new combination (supposed independant) is received, a new component is seen and $s$ is updated in consequence.

The decoding matrix size and delay must be kept under control, however seeing a component only indicates it will be decodable at one point in the future, but the delay is actually unbounded. For this reason, the receiver should not acknowledge every change of $s$ to the sender. For exemple, we can imagine a simplified critical scenario with a coding window $W$ of size 2: the receiver gets $P_1 \oplus P_2$, $P_1$ is seen, so it acknowledges $s = 2$. The sender removes $P_1$ from its coding window, adds $P_3$, and sends $P_2 \oplus P_3$. The receiver gets $P_2 \oplus P_3$, $P_2$ is seen, so it acknowledges $s = 3$. The process repeats while the sender has new data available, and the receiving buffer grows whereas no chuck can be decoded.

To solve this issue, as a security, the receiver infers the size of the coding windows $M$ from received combination lengths. $s$ is acknowledged to the sender in the the MPTCP connection-level data ACK field if and only if the number of undecoded combinations $m$ in the receiving buffer is less than $M$. If $m \geq M$, then $s + M - m - 1$ is acknowledged instead.

If components can be decoded in order after a reception event, the size of each one is read, the padding is removed and the data is copied to a receiving buffer then passed to the upper layer. These decoded components must be kept in memory to decode the next incoming combinations. However, the components that are decoded and not present in the incoming combinations anymore can be forgotten. The receiver must then pay attention to drop older (for instance delayed) combinations it will not be able to decode.

### 7.3.4 Feedback mechanism

The MPTCP `Data ACK` field performing global window acknowledgements now stores the `Next Seen Component Number`. It is now always 32 bits long and contains the next packet to be seen by the decoder. The sender gets the *next seen number* $s$ from the receiver and removes the segments $P_i$ where $i < s$. This feedback mechanism allows the coding window to move forward.

Note TCP ACKs mechanism at subflow level is kept unchanged.

## 7.4 Implementation details

We implemented an MPC-TCP in the Linux kernel by modifing the MPTCP reference implementation [42]. The main reason for chosing to develop a real-world implementation rather than

doing simulation is the impact of the cost of coding. Taking into account the delays induced by coding and decoding time in a network simulator is indeed a difficult task when implementing the protocol in a simulator, whereas it is obviously no concern with a real implementation.

The way adresses are advertised, subflows are created and destroyed is the same as MPTCP as defined in RFC 6824 [40]. However, four main modifications are made to the behaviour of MPTCP stack:

- The scheduler does not select the next segment from the window, instead, it calls a function to combine segments in the window.
- Combinations are always sent as a new mapping, requiring a new DSS option for each combination.
- At reception, received combinations are pushed to a decoding buffer implementing the gaussian elimination algorithm, and decoded segments are transmitted to the upper layer.
- The acknownledgements correspond to seen components rather than received segments.

To properly implement these modifications, we have to make a few changes to the headers.

### 7.4.1 Data Sequence Signal (DSS) option

The Data Sequence Signal (DSS) TCP option defined in MPTCP is not modified but the fields *Data ACK* and *Data sequence number* are interpreted differently. For consistency with MPTCP DSS option format, flag 0x02 must always be set to 0 and flag 0x08 must always be set to 1.

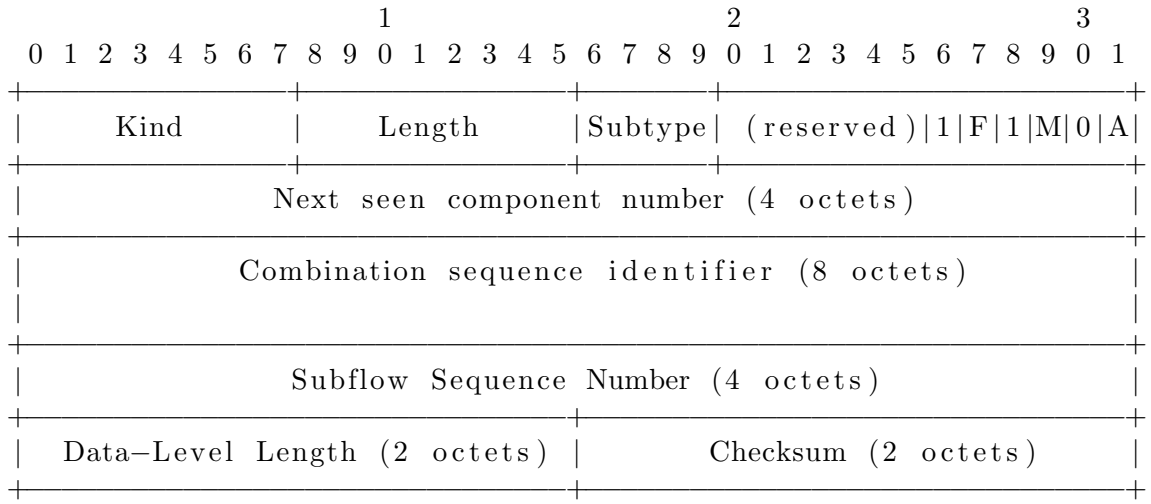



Figure 7.6: Reinterpreted Data Sequence Signal (DSS) option

#### 7.4.1.1 Combination sequence identifier

The MPTCP *Data sequence number* field is now the *Combination sequence identifier*. It is now always 64 bits long and contains an identifier describing the vector of coefficients used to compute the linear combination delivered in the TCP segment. The low order 32 bits correspond to the

first component number in the combination, the next 16 bits correspond to the combination length (i.e. components count) and the highest order 16 bits are a unique nonce. Figure 7.7 shows the structure in visual format.

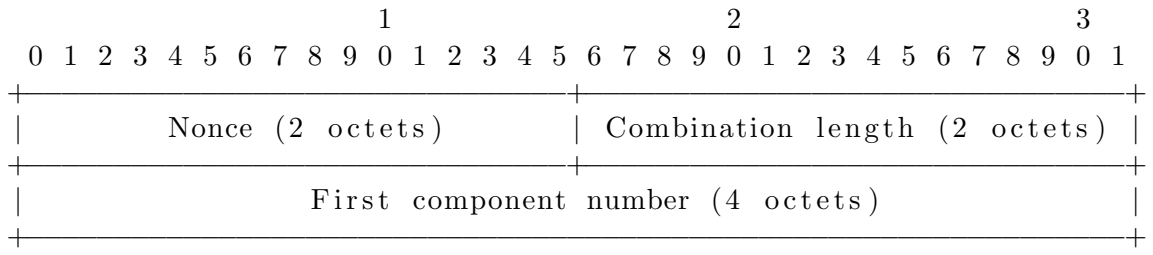



Figure 7.7: Combination sequence identifier structure

The nonce must be repeated as infrequently as possible on a single MPTCP connection. It should be set at an initial random value and incremented by 1 for each generated combination.

The sequence identifier can be used to retrieve the first component number and the combination length. This unambiguously gives the number of each component in the vector, since the components are contiguous. Then, it serves as a seed for Donald Knuth's MMIX 64-bits linear congruential generator to generate the coefficient for each component.

$$x_0 = \text{sequence\_identifier}$$
$$x_{i+1} = ax_i + c \mod 2^{64}$$

where $a = 6364136223846793005$ and $c = 1442695040888963407$

The successive coefficients are given by the 8 most significant bits of each $x_i$ $(i > 0)$ where these bits are not all null. Figure 7.8 shows a C implementation example.

```
uint64_t state = sequence;

uint8_t next_coefficient(void)
{
  while(true)
  {
    state = state*6364136223846793005ULL + 1442695040888963407ULL;
    uint8_t value = (uint8_t)(state >> 56);
    if(value) return value;
  }
}
```

Figure 7.8: MMIX 64-bits linear congruential generator adapted for Coded MPTCP

#### 7.4.1.2 Next seen component number

The MPTCP *Data ACK* field is now the *Next seen component number*. It is now always 32 bits long and contains the next packet to be seen by the decoder.

Even if TCP ACKs mechanism at subflow level is kept unchanged, we have modified the acknowledgement system at global level to keep track of seen packets. The sender side can the remove seen packets from its coding window, allowing to move it forward.

### 7.4.2 Component size

Components in the coding buffer are padded with zeros before coding. Before padding, the actual data chunk size is prepended as a 16-bit network byte order (big endian) unsigned integer. This 2-octet header is treated as the rest of the data during coding.

The decoder reads this integer once the component is decoded, then reads the data chunk according to its actual size, and deliver it to the upper layer.

## 7.5 Performance evaluation

### 7.5.1 Scenario

We evaluate the protocol in a typical head-of-line blocking scenario. Two links are set up with ns-3 emulation between two MPTCP-enabled virtual machines (Fig. 7.9). The client establishes a 2-path MPTCP connection (vanilla or coded) to the server, which then push data as fast as possible, and goodput is measured on the client at application level. Our actual setup is made of two User Mode Linux virtual machines with MPTCP and MPC-TCP kernel implementation whose traffic is routed through ns-3 running in realtime mode. For our tests, MPTCP is configured to use the round-robin scheduler.

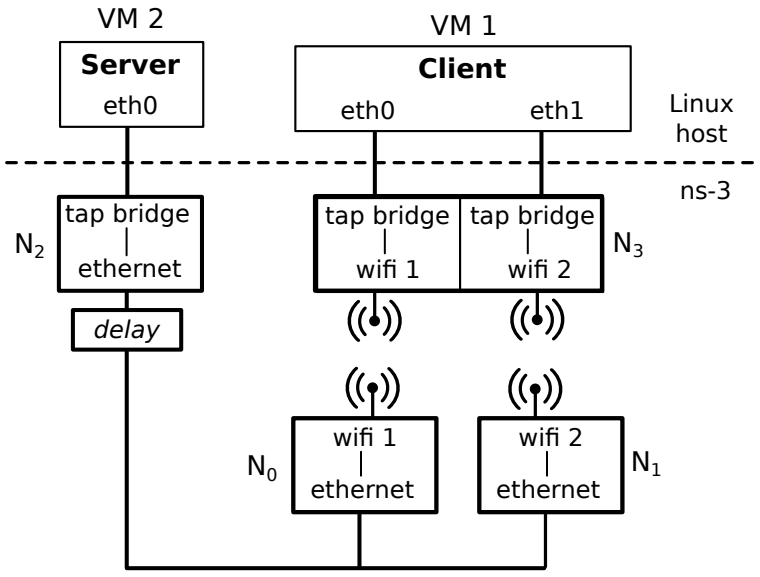



Figure 7.9: Emulated network for test scenario

On the second links, we change the token bucket capacity to zero repetetively to simulate an unreliable and flickering path, therefore the path is blocked for a fraction of the time. The link goes through cycles of 10s duration, and on each cycle, the link is working during the first part and blocked during the second part. For instance, if the loss blocking rate is 10%, the link

is working for 9s, then blocked for 1s, then the process repeats. Each transfer lasts 60s, and we report the average goodput over 10 runs.

### 7.5.2 Results

We create head-of-line blocking events by repetetively having a the second link dropping all traffic for 1 second intervals. We measure throughput given the period between these events. A fixed 100ms delay is added on server-side at link-level.

Figure 7.10 shows that MPC-TCP performs better than vanilla MPTCP when the second path becomes heavily unreliable, but worse when it works properly. Upon investigation, the worse performance comes from an overhead introduced by the coding scheme which tends to transmit redundant combinations, indicating the current feedback mechanism could be enhanced.

However, the really interesting aspect is that the goodput doesn't collapse even when the second path is blocked 50% of the time. It shows MPC-TCP is not affected by head-of-line blocking, as it can use the second path when it works, and is not affected when it stops working. On the contrary, with MPTCP vanilla, the second subflow prevent the other subflow from working properly as packets must be reinjected, so the working subflow is actualy not used at full capacity.

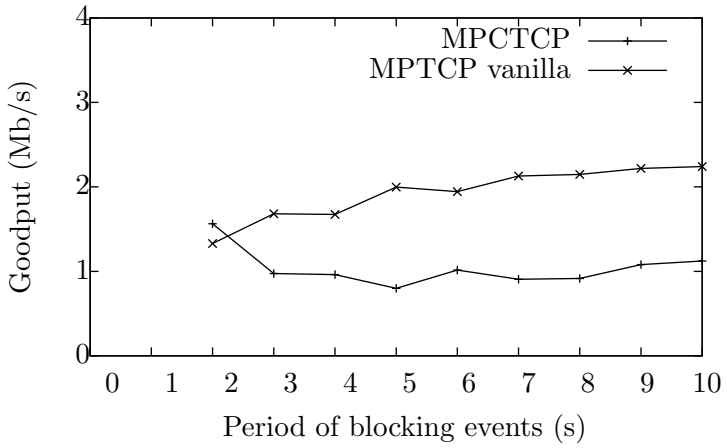



Figure 7.10: Goodput comparison given period of head-of-line-blocking events

## 7.6 Conclusion

In this section, we have addressed the issue of head-of-line blocking in MPTCP, and proposed MPC-TCP to solve it. In parallel, we demonstrate the feasibility of a simple and practical network coding implementation in MPTCP compatible with vanilla MPTCP and restrictive middleboxes.

As expected, it solves the head-of-line blocking problem by removing mapping issues at the price of added overhead, because of the simplistic feedback mechanism. We plan to fix it in order to make MPC-TCP a fully functional alternative for MPTCP, one approach to do so could

be dynamically switching the coding on or off depending on network circumstances to always get the best possible throughput.

# Chapter 8

# Conclusion

Intra-flow network coding is an interesting solution to enhance transmission reliability. Contrary to traditional forwarding where nodes transmit original packets without tampering with them, the core idea of intra-flow network coding is to forward combinations of the packets from the same flow rather than the packets themselves. Even if network coding seems to induce a tradeoff in terms of processing power as of today, this problem will easily be solved in the future with dedicated hardware.

In this work, we presented different contributions to show the potentials of intra-flow network coding to enhance reliability of data transfers over wireless networks.

Our first contribution concerns redundancy adaptation. We have derived a minimal redundancy bound to set the network coding redundancy according to the link quality and the targeted maximum application loss rate, indeed, we believe that a balance should be kept between application requirements and network overhead.

Based on this work, we proposed a distributed algorithm for redundancy adaptation in mesh networks. The algorithm enables to opportunistically make use of multiple available paths to route the coded packets to destination while offering optimized redundancy control, balancing network overhead and application needs.

We performed evaluations in order to compare this mechanism to common redundancy adaptation schemes. Our benchmark shows that our solution outperforms these schemes, particularly thanks to the possibility of controlling redundancy according to the application requirements.

We then extended this algorithm to take into account nodes willingness to take part in network coding operations. Since network coding involves a trade-off between network performance enhancement and coding operations cost, we allow each node to define its contributions to the coding operations. The node constraints are taken into account when forwarding the packet combinations while guaranteeing decoding at destination.

We evaluated the extended model as we did for its original version. The evaluation highlights how the coding operations are moved across the network while data delivery at destination is still guaranteed. If the actual number of coding nodes is low, this can result in a slightly increased overhead since more redundancy is necessary to offer the same guaranty, but the overhead is kept low enough.

Then, in our next contributions, we studied the interaction of network coding with TCP and its multipath extension MPTCP.

Network coding deployment with TCP raises concern about flow fairness. Indeed, TCP flows share the available bandwidth thanks to their congestion control algorithm, and mostly, the congestion signal is due to packet loss. Intra-flow coding hides losses independently of their cause, so congestion losses are hided from TCP, which can lead to unfairness situations.

We studied the impact of intra-flow network coding on fairness between coded TCP flows and non-coded TCP flows. To this end, we defined a dedicated fairness index to study the implications of running parallel coded and non coded TCP flows. The results show that unfairness happens in favor of coded flows but its impact is limited, because even with high redundancy factors a coded flow never starves a non-coded one. This encouraging observation indicates that congestion losses are correlated enough to cause decoding to fail and TCP to react properly to the congestion.

Then, we attempted two approaches to enhance MPTCP performance with intra-flow network coding. First, we investigated running MPTCP over network coding and then we implemented network coding directly in MPTCP.

We built an emulation setup to study the benefits of running MPTCP over network coding. Since MPTCP is based on TCP and relies on the same mechanisms, it is also subject to similar drawbacks, especially sensitivity to link loss. The tests carried out on our emulation system show that MPTCP goodput is significantly improved when running on pipeline coding, in particular, if intra-flow redundancy factor is properly tuned. This showcases the importance of redundancy adaptation, a challenge we addressed in the first part of this thesis.

MPTCP is specifically subject to head-of-line blocking. This happens when the destination is unable to accept data from one path because it is waiting data from another path and its buffer is full. Since network coding can help to solve this issue, we designed a practical MPTCP extension, MultiPath Coded TCP (MPC-TCP). The idea is, rather than running MPTCP over network coding, to implement network coding directly in MPTCP, while keeping a strict network retro-compatibility with TCP.

As expected, the solution solves the head-of-line blocking problem by removing mapping issues. However, the overhead is non-negligible, probably because of the current feedback mechanism. This should be fixed in a future work to make MPC-TCP a viable alternative to MPTCP. A potential solution could be to dynamically switch the coding on or off depending on network conditions when ensuring the best possible throughput.

The emulation platforms and tools that we developed for this work will be useful for our research team to carry out the investigations on network coding performances. In particular, it will allow to benchmark MPTCP in more realistic environments.

The platform should also serve to continue the work related to the integration of network coding with MPTCP. Some work is indeed still to be carried out to make MultiPath Coded TCP a viable alternative to MPTCP in realistic environments.

As a summary, this work allowed to explore the potential of intra-flow network coding through multiple issues. It has been the occasion to design network coding systems for different platforms, which has been very enriching, even if it has also been very time-consuming, since required implementations ranged from ns-3 emulation to Linux kernel. Even if inter-flow coding was not the focus of this work, it was still very interesting to learn the concepts of it, and its different applications.

Moreover, it has been the occasion to work with different teams, in particular within the

framework of the OPUS project. Collaboration with people from the industry has been really interesting, and the project provided a good opportunity to learn about different domains, such as video services, and to investigate possible deployments for Unmanned Aerial Vehicules (UAV), which need communication systems between them and with the ground.

Working with the Network Research Lab at UCLA has also been a great opportunity. The different encounters sharpened my interest in promising applications like autonomous cars, raising new challenges that can be adressed with network coding.

Since network coding implementations are not common and MPTCP is not yet implemented in simulators, this research has requiered an important work of implementation and testbed construction. For now, results are limited and should therefore be extented using the developped infrastructure.

Our adaptation algorithm can be applied to enhance video transmission in wireless mesh networks. It can be adapted to more specifically to provide variable content protection to video data. With SVC (Scalable Video Coding), video can be divided in subset bitstreams, which can be transported using an expanding window algorithm and intra-flow network coding, each window having a different protection using our redundancy bound.

To push the study further, investigating security issues linked to network coding would be particularly interesting. Network coding is indeed sensitive to pollution, because an invalid combination can corrupt entire groups of packets. More specifically, if a single invalid combination is present, generated combinations will all be invalid, and decoding will be impossible. Homomorphic signatures are a potential solution to this issues, but, even if enhancements such as Null keys exist, their very high cost can be prohibitive in most applications. For instance, using a smart algorithm for decoding, network nodes could take advantage of redundant combinations, which are normally useless, to detect pollution, and maybe eliminate it: when the system is inconsistent during decoding, a node could try to find the minimum set of combinations to remove, given the proper hypotheses.

Network coding could also be interesting as a tool to enhance distributed caching or storage. Nowadays, data storage is a growing challenge, so it would be very interesting to continue working in this direction.

# Appendix A

# Publications

## A.1 Collaborations

During the course of this work, I had the great opportunity to visit the Network Research Lab at UCLA. I was invited by Mario Gerla to work with his team about issues related to network coding, from january 2014 to april 2014 and from november 2014 to february 2015. This was the opportunity for a very interesting collaborative work, in particular about MPTCP.

It has also been the occasion to take part in the OPUS research project with industrial partners: Thales Communications, Vitec, and the startup Green Communications. The OPUS project focuses on network coding to enhance real-time video communications over wireless networks. It has been an opportunity to learn a lot on video processing, but also to present this work and exchange with people from the industry, which is really welcome in a research work.

## A.2 Articles

1. Paul-Louis Ageneau, Nadia Boukhatem and Mario Gerla, *Fairness Evaluation of Pipeline Coded and Non Coded TCP Flows*, IEEE International Conference on Communications (ICC), 2014

2. Paul-Louis Ageneau, Chuchu Wu, Nadia Boukhatem and Mario Gerla, *Redundancy Adaptation for Multi-Path Intra-Flow Network Coding in Wireless Mesh Networks*, IEEE Vehicular Technology Conference (VTC2016-Fall), 2016

3. Paul-Louis Ageneau and Nadia Boukhatem, *Multipath TCP over Network Coding for Wireless Networks*, IEEE Annual Consumer Communications and Networking Conference (CCNC), 2017

4. Paul-Louis Ageneau, Nadia Boukhatem and Mario Gerla, *Practical Random Linear Coding for MultiPath TCP: MPC-TCP*, International Conference on Telecommunications (ICT) 2017

5. Paul-Louis Ageneau, Nadia Boukhatem and Mario Gerla, *Constraint-Aware Multi-Path Intra-Flow Network Coding in Wireless Mesh Networks*, International Wireless Communications and Mobile Computing Conference (IWCMC), 2017

6. Hana Baccouch, Paul-Louis Ageneau, Nicolas Tizon, Nadia Boukhatem, *Prioritized Network Coding Scheme For Multi-Layer Video Streaming*, 7th International Conference On Network Of the Future, 2016

7. Hana Baccouch, Paul-Louis Ageneau, Nicolas Tizon and Nadia Boukhatem, *Prioritized Network Coding Scheme For Multi-Layer Video Streaming*, IEEE Annual Consumer Communications and Networking Conference (CCNC), 2017

8. Hana Baccouch, Paul-Louis Ageneau, Nicolas Tizon and Nadia Boukhatem, *Network Coding Schemes For Multi-Layer Video Streaming On Multi-Hop Wireless Networks*, IEEE Wireless Communications and Networking Conference (WCNC), 2017

9. Hana Baccouch, Paul-Louis Ageneau, Nicolas Tizon, Nadia Boukhatem and Thi-Mai-Trang Nguyen, *Bounded Network Coding Redundancy for Multi-Layer Video Streaming*, International Wireless Communications and Mobile Computing Conference (IWCMC), 2017

## A.3 Deliverables

1. Paul-Louis Ageneau, Thuong Van Vu, Nadia Boukhatem, Michel Bourdellès, Alexandre Laube, Khaldoun Al Agha, *OPUS L2.1: Methods for optimizing network communications*

2. Hana Baccouch, Paul-Louis Ageneau, Nadia Boukhatem, Michel Bourdellès and Khaldoun Al Agha, *OPUS L2.3: First results of communications optimization simulations*

# Appendix B

# Redundancy bound derivation

## B.1 Chernoff bound

We use the Chernoff bound method to get an upper bound on the relative error for the binomal distribution $X = \sum_{k=1}^{n} X_k$ with $\forall k \in \{1, ..., n\}, X_k \in \{0, 1\}$ and $P(X_k = 1) = p$, *i.e.* a bound to $P(X > \lambda\mu)$ with $\mu = E[X]$ and $\lambda \geqslant 1$. The moment-generating function of $X_k$ is, for $t > 0$:

$$M_{X_k}(t) = E[e^{tX_i}] = pe^t + (1-p)e^0 = 1 + p(e^t - 1)$$

So we get the following bound:

$$M_{X_k}(t) \leqslant e^{p(e^t-1)}$$

And, for the moment-generating function of $X$:

$$M_X(t) = \prod_{k=1}^{n} M_{X_k}(t) \leqslant \prod_{k=1}^{n} e^{p(e^t-1)}$$

$$M_X(t) \leqslant \exp\left(\sum_{k=1}^{n} p(e^t - 1)\right) = e^{(e^t-1)E[X]} = e^{(e^t-1)\mu}$$

Let's introduce $\lambda > 1$ and compute the probability for the number of losses to be more than $\lambda\mu$:

$$P(X > \lambda\mu) = P(e^{tX} > e^{t\lambda\mu})$$

As a reminder, Markov's inequality for any positive random variable Y states:

$$\forall a > 0 \;\; P(Y \geqslant a) \leqslant \frac{E[Y]}{a}$$

Applying Markov's inequality for $Y = e^{tX}$ gives us:

$$P(X > \lambda\mu) \leqslant \frac{E[e^{tX}]}{e^{t\lambda\mu}} = \frac{M_X(t)}{e^{t\lambda\mu}} \leqslant \frac{e^{(e^t-1)\mu}}{e^{t\lambda\mu}}$$

Let's choose $t = \ln\lambda > 0$, we eventually get the following Chernoff bound:

$$P(X > \lambda\mu) \leqslant \left(\frac{e^{\lambda-1}}{\lambda^\lambda}\right)^\mu$$

Given this inequality is trivially true for $\lambda = 1$, it can be extended to $\lambda \geqslant 1$.

If we choose $\lambda = \frac{r-1}{pr}$, $\lambda \geqslant 1$ since $r \geqslant r_{\min}(p) = \frac{1}{1-p}$ we have:

$$\lambda\mu = \frac{r-1}{pr} \times prm = (r-1)m$$

And we eventually get the following upper bound for the probability to lose a given generation:

$$P(X > (r-1)m) \leqslant \exp\left(-prm(1 - \lambda + \lambda \ln \lambda)\right)$$

## B.2 Bound on $r$

The redundancy $r$ must verify:

$$\exp\left(-\frac{1}{2}mpr\left(\frac{r-1}{pr} - 1\right)^2\right) \leq \tau$$

This gives, with $K = -\frac{\ln \tau}{m}$:

$$pr\left(\frac{r-1}{pr} - 1\right)^2 \geq 2K$$

$$(1-p)^2 r^2 - 2\left((1-p) + Kp\right) r + 1 \geq 0$$

$$\Delta = 4Kp(Kp + 2(1-p)) > 0$$

With the constraint $r \geq r_{\min} = \frac{1}{1-p}$, the solution of the inequation is:

$$\begin{aligned} r \geq r_0 &= \frac{(1-p) + Kp + \sqrt{Kp}\sqrt{Kp + 2(1-p)}}{(1-p)^2} \\ &= \frac{1}{1-p}\left(1 + \frac{Kp}{1-p}\left(1 + \sqrt{1 + 2\frac{1-p}{Kp}}\right)\right) \end{aligned}$$

We finally obtain the redundancy lower bound:

$$r \geq \frac{1}{1-p}\left(1 + C\left(1 + \sqrt{1 + \frac{2}{C}}\right)\right)$$

$$\text{where } C = \frac{Kp}{1-p}$$

Given the actual parameters values (in particular $p << 1$), $C << 1$, so $\frac{1}{C} >> 1$ and $1 + \frac{2}{C} \simeq \frac{2}{C}$, and we get the approximation:

$$r \geq r_{\text{bound}}(m, p, \tau) = \frac{1}{1-p}\left(\frac{1 + (1 + \sqrt{2C})^2}{2}\right)$$

$$\text{where } C = \frac{-\ln \tau}{m}\frac{p}{1-p}$$

# List of Figures


2.1 Butterfly topology without (left) and with (right) network coding . . . . . 18

2.2 Multicast without (left) and with (right) network coding . . . . . 21

2.3 Network coding redundancy example with 3 original packets and 4 combinations (redundancy factor $r = 4/3$) . . . . . 22

2.4 Batch coding principle with a generation size $m = 3$ and one redundant combination 23

2.5 Pipeline coding principle with a generation size $m = 3$ and one redundant combination . . . . . 23

2.6 Partial decoding highlighting seen packets . . . . . 24

2.7 Opportunistic routing principle: neighbor node R2 can also serve as a potential relay, adding possible paths. . . . . 25

2.8 Opportunistic routing advantage: overall packet loss is reduced when multiple relays are available . . . . . 26

3.1 Redundancy control for network coding taxonomy . . . . . 36

3.2 Generation loss probability upper bound given the redundancy factor $r$ for different values of generation size $m$ ($p = 0.1$) . . . . . 38

3.3 Redundancy factor $r_{\text{bound}}$ given the generation loss probability upper bound $\tau$ for different values of generation size $m$ ($p = 0.1$) . . . . . 39

3.4 Node $i$ overhears a combination from $k$ and sends a new combination. . . . . 40

3.5 Measured generation loss probability given the redundancy factor $r$ for different values of generation size $m$ ($p = 0.1$) . . . . . 43

3.6 Topologies with different numbers of lines of relays with link loss $p$ between relays 44

3.7 Generation loss at destination given relay loss $p$ with one line of relay nodes . . . 45

3.8 Relative overhead given relay loss $p$ with one line of relay nodes . . . . . 45

3.9 Relative overhead given network relay loss $p$ with two lines of relay nodes . . . . 46

3.10 Relative overhead given relay loss $p$ with three lines of relay nodes . . . . . 46

3.11 Generation loss at destination in different topologies given the fraction of non-coding nodes . . . . . 47

3.12 Network overhead in different topologies given the fraction of non-coding nodes . 47

3.13 Measured generation loss probability for on-the-fly and delayed recoding given the number of hops on the path between source and destination ($p = 0.1, \tau = 0.1\%$) 48

4.1 Nodes $i$ and $i'$ relay traffic with different contributions. . . . . 51
4.2 Simulated network topologies with 2 or 3 lines . . . . . 55
4.3 Effect of $\alpha$ factor on first line . . . . . 55
4.4 Relative overhead given $\alpha$ factor on first line . . . . . 56
4.5 Grid topology with $n \times m$ nodes . . . . . 56
4.6 Relative overhead given fraction of nodes with $\alpha = 0.1$ . . . . . 57

5.1 TCP over pipeline coding . . . . . 62
5.2 Simulated bottleneck topology . . . . . 63
5.3 Congestion window with (top) and without (bottom) coding . . . . . 64
5.4 Simulated bottleneck topology. All links have 1 ms delay and uniform loss with probabilty $p$. . . . . 65
5.5 Goodput comparison between concurrent flows, one with coding at different coding redundancy factors and one without ($p = 1\%$) . . . . . 65
5.6 Goodput comparison between concurrent flows, one with coding at different generation sizes and one without ($r = 1.25$, $p = 1\%$) . . . . . 66
5.7 Goodput comparison between concurrent flows at different loss rates ($g = 16$, $r = 1.25$) . . . . . 66
5.8 Comparison of capacity utilization on a lossy link between a first case with a non-coded and a coded flow, and a second case with two non-coded flows . . . . 67
5.9 Formalization of capacity utilization on a lossy link between allocation $A$ and allocation $A'$ where coded flow are replaced with non-coded ones . . . . . 68
5.10 Jain's index at different coding factors for a coded and a non-coded flow ($p = 1\%$) 70
5.11 Jain's index at different loss rates for a coded ($g = 16$, $R = 1.25$) and a non-coded flow . . . . . 71

6.1 MPTCP with two subflows running over intra-flow pipeline coding . . . . . 74
6.2 Network emulation setup with ns-3 and two User-Mode Linux virtual machines . 75
6.3 ns-3 emulation internal details . . . . . 76
6.4 Pipeline coding as implemented in ns-3 nodes . . . . . 76
6.5 Application goodput for MPTCP on two links given loss rate $p$ with 100ms RTT (generation size 8) . . . . . 77
6.6 Application goodput given loss rate comparison given redundancy factor $r$ (generation size 16) . . . . . 77

6.7 Application goodput given redundancy factors $r$ for loss rate $p = 0.1$ (generation size 8) . . . 78
6.8 Application goodput for MPTCP given loss rate $p$ for different generation sizes (RTT $= 100\text{ms}, r = 1.50$) . . . 78
6.9 Application goodput for MPTCP without coding given loss rate $p$ according to RTT . . . 79
6.10 Application goodput for MPTCP with coding given loss rate $p$ according to RTT ($r = 1.5$) . . . 79

7.1 Head-of-line blocking issue encountered with MPTCP . . . 81
7.2 Simplified principle of coding over MPTCP . . . 82
7.3 Head-of-line blocking issue encountered with MPTCP . . . 83
7.4 Head-of-line blocking solution with network coding . . . 83
7.5 Overview of the coding/decoding principle with two subflows . . . 85
7.6 Reinterpreted Data Sequence Signal (DSS) option . . . 88
7.7 Combination sequence identifier structure . . . 89
7.8 MMIX 64-bits linear congruential generator adapted for Coded MPTCP . . . 89
7.9 Emulated network for test scenario . . . 90
7.10 Goodput comparison given period of head-of-line-blocking events . . . 91

# Bibliography


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

[2] G. Xylomenos, G.C. Polyzos, P. Mahonen, and M. Saaranen. TCP performance issues over wireless links. *Communications Magazine, IEEE*, 39(4):52–58, 2001.

[3] Alan Ford, Costin Raiciu, Mark Handley, Olivier Bonaventure, et al. TCP Extensions for Multipath Operation with Multiple Addresses: draft-ietf-mptcp-multiaddressed-03. Technical report, Roke Manor, 2011.

[4] Thuong Van Vu. *Application of network coding in wireless networks : coding conditions and adaptive redundancy control*. Theses, Université Pierre et Marie Curie - Paris VI, April 2014.

[5] S. Y.R. Li, R. W. Yeung, and Ning Cai. Linear network coding. *IEEE Trans. Inf. Theor.*, 49(2):371–381, February 2003.

[6] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *Information Theory, 2003. Proceedings. IEEE International Symposium on*, pages 442–, June 2003.

[7] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: Practical wireless network coding. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 243–254, New York, NY, USA, 2006. ACM.

[8] M. Luby. Lt codes. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 271–280, 2002.

[9] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, June 2006.

[10] Chien-Chia Chen, Soon Y. Oh, Phillip Tao, Mario Gerla, and M. Y. Sanadidi. Pipeline network coding for multicast streams. In *Proceedings of the International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2010)*, Seattle, USA, April 2010.

[11] J.K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros. Network coding meets TCP. In *INFOCOM 2009, IEEE*, pages 280–288, 2009.

[12] N. Chakchouk. A survey on opportunistic routing in wireless communication networks. *IEEE Communications Surveys Tutorials*, 17(4):2214–2241, Fourthquarter 2015.

[13] Sanjit Biswas and Robert Morris. ExOR: Opportunistic Multi-hop Routing for Wireless Networks. *SIGCOMM Comput. Commun. Rev.*, 35(4):133–144, August 2005.

[14] Douglas SJ De Couto. *High-throughput routing for multi-hop wireless networks*. PhD thesis, Citeseer, 2004.

[15] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. MORE: A Network Coding Approach to Opportunistic Routing. 2006.

[16] Yunfeng Lin, B. Li, and Ben Liang. Codeor: Opportunistic routing in wireless mesh networks with segmented network coding. In *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*, pages 13–22, Oct 2008.

[17] Y. Lin, B. Liang, and B. Li. Slideor: Online opportunistic network coding in wireless mesh networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5, March 2010.

[18] Pablo Garrido, David Gómez, Ramón Agüero, and Joan Serrat. Combination of Intra-flow Network Coding and Opportunistic Routing: Reliable Communications over Wireless Mesh Networks. In *8th International Conference on Simulation Tools and Techniques*, SIMUTools 2015, pages 191–199. ICST, 2015.

[19] X. Zhang and B. Li. Optimized multipath network coding in lossy wireless networks. *IEEE Journal on Selected Areas in Communications*, 27(5):622–634, June 2009.

[20] Joon-Sang Park, Mario Gerla, Desmond S. Lun, Yunjung Yi, and Muriel Médard. Codecast: a network-coding-based ad hoc multicast protocol. *Wireless Communications, IEEE*, 13(5):76–81, 2006.

[21] Chien-Chia Chen, Clifford Chen, Soon Y. Oh, Joon-Sang Park, Mario Gerla, and M.Y. Sanadidi. ComboCoding: Combined intra-/inter-flow network coding for TCP over disruptive MANETs. *Journal of Advanced Research*, 2(3):241–252, 2011.

[22] Chien-Chia Chen, Joon-Sang Park, M. Y. Sanadidi, Guruprasad Tahasildar, Yu-Ting Yu, and Mario Gerla. CodeMP: Network coded multipath to support TCP in disruptive MANETs. In *Proceedings of the 2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*, MASS '12, pages 209–217, Washington, DC, USA, 2012. IEEE Computer Society.

[23] H. Seferoglu, A. Markopoulou, and K.K. Ramakrishnan. I2NC: Intra- and inter-session network coding for unicast flows in wireless networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1035–1043, April 2011.

[24] Hamlet Medina Ruiz, Michel Kieffer, and Batrice Pesquet-Popescu. An adaptive redundancy scheme for tcp with network coding. In *IEEE International Symposium on Network Coding (NETCOD), United States. IEEE.* Citeseer, 2012.

[25] D. Koutsonikolas, C. C. Wang, and Y. C. Hu. Ccack: Efficient network coding based opportunistic routing through cumulative coded acknowledgments. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[26] Thuong Van Vu, Nadia Boukhatem, Thi Mai Trang Nguyen, and Guy Pujolle. Dynamic coding for tcp transmission reliability in multi-hop wireless networks. In *WoWMoM*, pages 1–6, 2014.

[27] Zhen Yu, Yawen Wei, B. Ramkumar, and Yong Guan. An Efficient Signature-Based Scheme for Securing Network Coding Against Pollution Attacks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[28] Jingyao Zhang and Pingyi Fan. On network coding in wireless ad-hoc networks. In *2005 2nd Asia Pacific Conference on Mobile Technology, Applications and Systems*, pages 8 pp.–8, Nov 2005.

[29] Chuchu Wu, Mario Gerla, and Mihaela van der Schaar. Social norm incentives for secure network coding in MANETs. In *International Symposium on Network Coding (NETCOD)*, pages 179–184, 2012.

[30] Tingting Chen and Sheng Zhong. Inpac: An enforceable incentive scheme for wireless networks using network coding. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[31] Tingting Chen, Ankur Bansal, and Sheng Zhong. A reputation system for wireless mesh networks using network coding. *J. Netw. Comput. Appl.*, 34(2):535–541, Mar 2011.

[32] Minji Kim, Muriel Médard, and João Barros. Modeling network coded TCP throughput: a simple model and its validation. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '11, pages 131–140, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[33] N. K G Samaraweera. Non-congestion packet loss detection for TCP error recovery using wireless links. *Communications, IEE Proceedings*, 146(4):222–230, 1999.

[34] Fabio Martignon and Luigi Fratta. Loss differentiation schemes for TCP over wireless networks. In *Proceedings of the Third international conference on Quality of Service in Multiservice IP Networks*, QoS-IP'05, pages 586–599, Berlin, Heidelberg, 2005. Springer-Verlag.

[35] C. P. Fu and S. C. Liew. TCP Veno: TCP Enhancement for Transmission Over wireless Access Networks. pages 216–227, February 2003.

[36] E. H.-K. Wu and Mei-Zhen Chen. JTCP: jitter-based TCP for heterogeneous wireless networks. *IEEE J.Sel. A. Commun.*, 22(4):757–766, September 2006.

[37] Hamlet Medina Ruiz, Michel Kieffer, and Béatrice Pesquet-Popescu. An adaptive redundancy scheme for tcp with network coding. In *IEEE International Symposium on Network Coding (NETCOD)*, United States, 2012.

[38] MinJi Kim, Jason Cloud, Ali ParandehGheibi, Leonardo Urbina, Kerim Fouli, Douglas J. Leith, and Muriel Médard. Network coded TCP (CTCP). *CoRR*, abs/1212.2291, 2012.

[39] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984.

[40] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[41] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 99–112, Berkeley, CA, USA, 2011. USENIX Association.

[42] Christoph Paasch, Sébastien Barré, Olivier Bonaventure, Gregory Detal, and Fabien Duchêne. Multipath TCP in the Linux Kernel. `http://www.multipath-tcp.org`.

[43] Jason Cloud, Flavio Pin du Calmon, Weifei Zeng, Giovanni Pau, Linda Zeger, and Muriel Medard. Multi-Path TCP with Network Coding for Mobile Devices in Heterogeneous Networks. June 2013.

[44] Ming Li, A. Lukyanenko, and Yong Cui. Network coding based multipath TCP. In *2012 IEEE Conference on Computer Communications Workshops (INFOCOM)*, pages 25–30, March 2012.

[45] C. Xu, Z. Li, L. Zhong, H. Zhang, and Gabriel-Miro Muntean. CMT-NC: Improving the Concurrent Multipath Transfer Performance using Network Coding in Wireless Networks. *Vehicular Technology, IEEE Transactions on*, PP(99):1–1, 2015.

[46] N. Arianpoo, I. Aydin, and V.C.M. Leung. Network coding: A remedy for receiver buffer blocking in the concurrent multipath transfer of data over multi-hop wireless networks. In *Communications (ICC), 2014 IEEE International Conference on*, pages 3258–3263, June 2014.

# Résumé

## 1 Introduction

Depuis le début des années 1960, les réseaux informatiques, ainsi que leurs usages, ont drastiquement évolués, alors que les principes de transmission de paquets sont restés fondamentalement les mêmes.

Bien qu'aujourd'hui la majorité des accès se fassent par des technologies sans fil, les paquets sont en général toujours transmis d'un nœud à l'autre comme des blocs de données inaltérables. Cependant, ce paradigme fondamental a récemment été remis en question par de nouvelles techniques comme le codage réseau, qui promet des améliorations conséquentes des performances pour peu que les nœuds puissent mixer les paquets ensemble.

Les réseaux sans fil sont très répandus aujourd'hui, en particulier depuis que les normes WiFi 802.11 et 3G/4G/LTE sont devenus les principales technologies de connexion des appareils auprès du grand public. À l'avenir, les appareils mobiles auront besoin de se connecter de partout, les smartphones génèreront davantage de trafic, beaucoup d'objets de la vie courante seront raccordés aux réseaux mobiles, les voitures et les drones auront besoin de communications de bonne qualité entre eux et avec des stations de base.

Cependant, les réseaux sans fil souffrent de taux de perte de paquets importants à cause des obstacles, des interférences et des mouvements. La qualité d'un lien peut se dégrader rapidement et causer des pertes de paquets. Les protocoles de la couche MAC implémentent des retransmissions pour pallier en partie ce problème, mais des pertes peuvent toujours avoir lieu pour les couches protocolaires plus hautes après un certain nombre de retransmissions échouées.

Ces problèmes tendent à empirer dans le cas des réseaux sans fil de type maillé avec des nœuds jouant le rôle de relais. L'augmentation du nombre de relais non seulement augmente les interférences, mais le nombre de sauts à franchir augmente aussi la perte subie à destination. Dans de telles topologies, les protocoles MAC peuvent s'avérer insuffisants pour empêcher des pertes potentielles et ainsi garantir une fiabilité suffisante pour la plupart des applications.

# 2 Contexte et définitions

## 2.1 Définitions et bénéfices du codage réseau

Le déploiement du codage réseau dans les réseaux sans fil est le sujet d'une attention croissance de la part de chercheurs depuis son introduction. Le codage réseau permet d'améliorer le débit et la fiabilité des transmissions. De plus, les réseaux sans fil, du fait de leur transmission par diffusion et des possibilités d'écoute, sont considérés comme des environnements particulièrement adaptés son application.

Le codage réseau désigne une famille de techniques basées sur un principe fondamental : au lieu de transmettre les paquets un par un , les nœuds du réseau peuvent combiner plusieurs paquets pour créer un paquet *codé*, aussi appelé *combinaison*, avant de la transmettre. Ce processus de combinaison de paquets est appelé *codage*, et le processus permettant de retrouver les paquets originaux à partir des paquets codés est appelé *décodage*.

Ces techniques peuvent être considérées comme une alternative à la transmission traditionnelle des paquets dans un réseau. En fonction des applications, les bénéfices sont multiples : le nombre de transmissions nécessaires pour transmettre des données dans un réseau peut être réduit, les pertes de paquets peuvent être prévenues et la fiabilité des flux peut être améliorée.

Par exemple, le codage réseau a été proposé pour maximiser la capacité du réseau, c'est-à-dire le trafic maximal que le réseau peut transmettre des sources vers les destinations. La topologie de type "papillon" (figure 1) illustre un cas où le codage réseau fonctionne nettement mieux que la retransmission de paquets. Elle consiste en deux noeuds source transmettant deux flux A et B à travers un réseau avec trois chemins vers deux noeuds de destination. Chaque noeud de destination souhaite recevoir les deux flux, et chaque lien a une capacité d'un paquet par unité de temps. La capacité du réseau est de 3 paquets par unité de temps avec la retransmission traditionnelle, mais de 4 paquets par unité de temps avec le codage réseau.

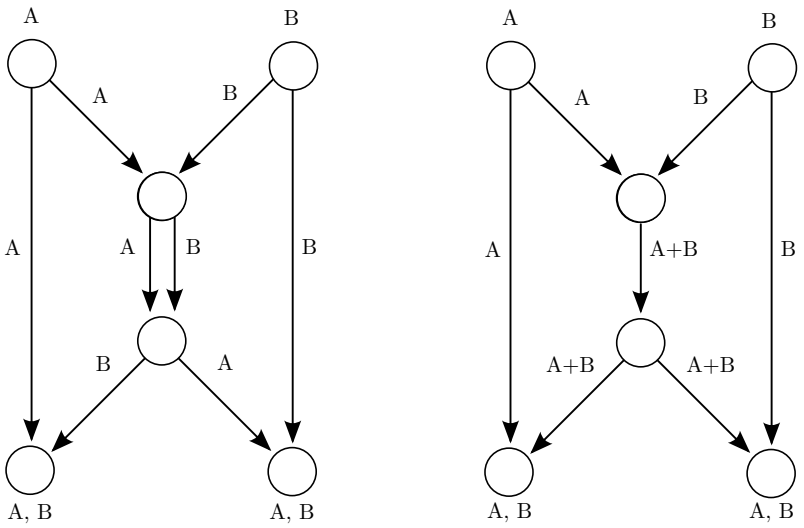



FIGURE 1 – Topologie de type "papillon" sans (à gauche) et avec codage réseau (à droite)

## 2.2 Le codage réseau linéaire

La grande majorité des travaux appliqués sur le sujet utilisent le codage réseau linéaire. Le codage linéaire utilise l'algèbre linéaire pour obtenir le codage des paquets : les paquets sont considérés comme des vecteurs et les combinaisons de paquets sont des combinaisons linéaires. Il a été prouvé que le codage linéaire est suffisant pour obtenir un débit optimal dans les situations de multidiffusion.

Les coefficients utilisés pour calculer les combinaisons peuvent être obtenus de manière déterministe ou aléatoire. Il a été démontré que les coefficients aléatoires pour le codage réseau linéaire permettent presque d'atteindre la capacité du réseau dans les systèmes de transmission de diffusion comme les réseaux sans fil en utilisant un algorithme décentralisé, à condition que la taille du champ soit suffisamment grande.

Plus la taille du champ est grande, plus la probabilité pour le récepteur est d'obtenir des combinaisons linéairement indépendantes. En pratique, un champ de $2^8$ éléments est utilisé la plupart du temps car il suffit d'obtenir une bonne probabilité de décodage, et il permet de meilleures performances, car les symboles sont alors des octets.

## 2.3 Approches du codage réseau

Compte tenu des flux à partir desquels les paquets sont codés ensemble, nous pouvons distinguer deux catégories de techniques du codage réseau : le codage inter-flux et le codage intra-flux. Le codage inter-flux a pour objectif d'atteindre la capacité maximale du réseau dans les topologies où la transmission conventionnelle de paquets ne le permet pas, en codant ensemble les paquets de différents flux passant à travers un même nœud. Le codage intra-flux peut soit servir à atteindre la capacité maximale pour les flux multicasts, soit à augmenter la fiabilité des flux unicast sur des réseaux non fiables, en combinant des paquets appartenant au même flux.

### 2.3.1 Codage inter-flux

Le codage réseau inter-flux vise à maximiser la capacité du réseau en combinant des paquets provenant de différents flux unicast dans les relais.

Les paquets provenant de différents flux sont combinés afin d'envoyer moins de paquets et de réduire le temps de transmission sans fil. L'idée est de ne transmettre que les informations nécessaires. Le temps de transmission économisé est disponible pour d'autres flux. Il en résulte un débit accru pour chaque flux et une capacité de réseau accrue. Le réseau de papillons présenté dans la section précédente (figure 1) montre un exemple classique de la façon dont le débit disponible peut être augmenté.

### 2.3.2 Codage intra-flux

Le codage réseau intra-flux, sujet principal de ce travail, permet d'augmenter la fiabilité d'un flux en combinant les paquets de celui-ci ensemble. L'idée est d'envoyer des combinaisons de paquets sortants à partir du même flux plutôt que d'envoyer les paquets eux-mêmes.

En effet, le codage réseau fournit des avantages par rapport aux retransmissions. Les mécanismes de retransmission conventionnels, par exemple Automatic Repeat Query (ARQ), nécessitent de

déterminer si chaque paquet est correctement reçu ou perdu, afin d'effectuer une retransmission de ce paquet particulier. Avec le codage réseau, la source peut combiner les paquets à transmettre ensemble, alors la destination doit uniquement avoir des combinaisons suffisantes pour décoder les informations. Ainsi, la source ne se préoccupe pas des données spécifiques à retransmettre.

À destination, un pivot de Gauss permet de récupérer tous les paquets pourvu qu'assez de combinaisons linéaires indépendantes aient été reçues. Selon la mise en œuvre du codage, les noeuds intermédiaires peuvent soit recoder les paquets en combinant les combinaisons transférées, soit simplement les transférer.

Pour compenser les pertes, plus de combinaisons que les paquets d'origine peuvent être générées. À destination, les paquets d'origine peuvent être décodés, à condition que des combinaisons indépendantes soient reçues. Les données d'origine peuvent être récupérées même si certains paquets sont perdus.

De plus, cette technique peut être avantageusement exploitée lorsque les noeuds peuvent écouter les paquets qui ne leur sont pas destinés et effectuer des retransmissions opportunistes. Avec des retransmissions traditionnelles, les nœuds voisins devraient échanger des informations explicitement pour effectuer une retransmission, car un paquet manquant à destination ne peut être récupéré qu'en retransmettant ce même paquet. Le codage réseau peut résoudre ce problème de manière transparente car toute combinaison innovante peut être utilisée pour décoder les paquets d'origine.

## 2.4 Méthodes de mise en œuvre du codage intra-flux

### 2.4.1 Redondance du codage réseau

Le codage réseau intra-flux permet d'envoyer plus de combinaisons que les paquets d'origine afin d'accroître la fiabilité du flux. Ainsi, un flux peut supporter un certain nombre de pertes de paquets tout en transmettant les données d'origine. Le rapport entre le nombre de combinaisons envoyées et le nombre de paquets originaux, appelé facteur de redondance, est un paramètre critique.

Le facteur de redondance devrait être suffisamment grand pour compenser les pertes de liens et garantir la réception de suffisamment de combinaisons à la réception pour s'assurer que les paquets sont décodés. Cependant, une valeur trop élevé peut entraîner une surcharge inutile du réseau.

Par exemple, si l'expéditeur veut envoyer 3 paquets, il peut envoyer 4 combinaisons afin de protéger les paquets d'éventuelles pertes (figure 2). Dans ce cas, le facteur de redondance $r$ est de 4/3. Si l'une des combinaisons est perdue, la destination peut encore décoder les paquets d'origine sans aucune différence. Notez que si aucune des combinaisons n'est perdue, la destination ignore simplement la combinaison surnuméraire car elle n'apporte pas de nouvelles informations et est inutile pour le processus de décodage. Une telle combinaison s'appelle *non-innovant.*

### 2.4.2 Codage à générations

Pour implémenter le codage réseau, il est nécessaire d'identifier quels paquets coder ensemble. L'une des approches les plus connues s'appelle le codage à générations.

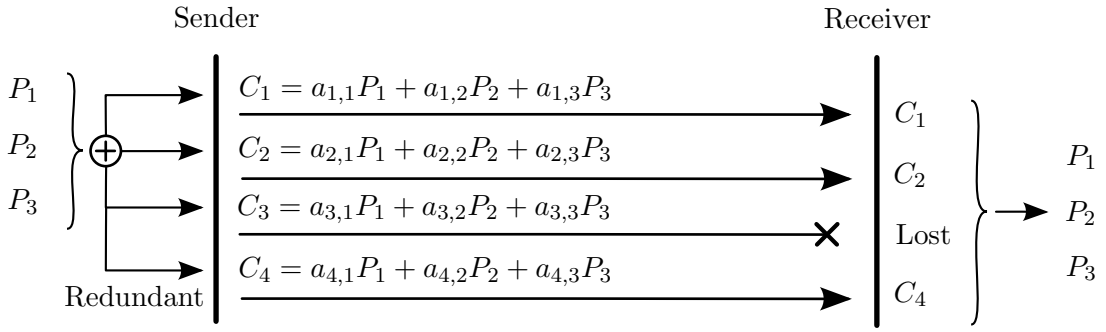



FIGURE 2 – Exemple de redondance du codage réseau avec 3 paquets d'origine et 4 combinaisons (facteur de redondance $r = 4/3$)

La quantité de calculs nécessaire au codage et, plus particulièrement, au décodage, dépend principalement du nombre de paquets codés ensemble. En raison de contraintes techniques, dans les implémentations pratiques, une technique commune consiste à grouper des paquets dans des lots consécutifs appelés générations. Les paquets appartenant à la même génération sont codés ensemble.

Les petites générations permettent de réduire le temps et le délai de traitement. Cependant, ils réduisent également l'efficacité du codage réseau. La raison principale est évidente : la destination nécessite suffisamment de paquets d'une génération spécifique pour effectuer le décodage de cette génération. Par conséquent, les générations plus petites ont tendance à souffrir du même inconvénient que le transfert de paquets. L'expédition traditionnelle peut être considérée comme une génération de taille 1.

Selon la manière dont les packets sont codés ensembles dans une génération, on peut distinguer deux types de méthodes.

Dans le Batch coding, ou codage par lots, la source combine les paquets de la même génération ensemble (figure 3).

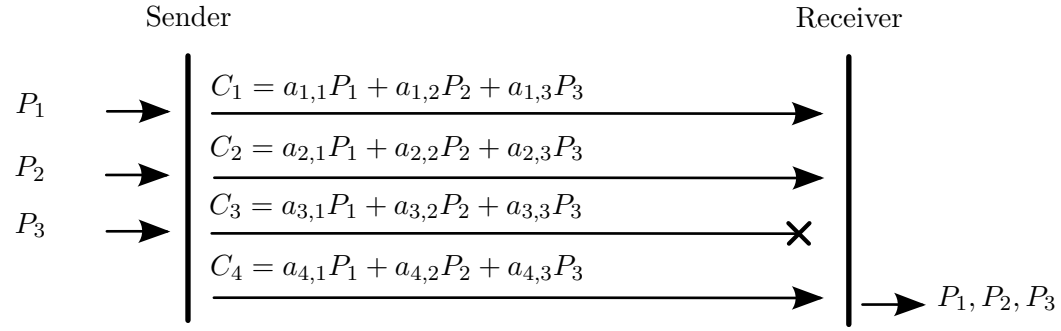



FIGURE 3 – Principe du Batch coding avec une taille de génération $m = 3$ et une combinaison redondante

À l'inverse, le Pipeline coding est un système du codage réseau à générations permettant un délai plus faible. Il est particulièrement adapté aux sessions interactives et multimédias en temps réel. En outre, il fonctionne bien lorsqu'il est combiné avec des applications basées sur protocole TCP. Le Pipeline coding code et décode les paquets progressivement. Son principe est de stocker les paquets sortants dans un tampon de codage et d'en envoyer des combinaisons dès que possible. Lorsqu'un nouveau paquet arrive à partir de l'application, il est ajouté au tampon de codage et une ou plusieurs combinaisons (en moyenne $r$, où $r$ est le facteur de redondance)

est immédiatement envoyé (figure 4).

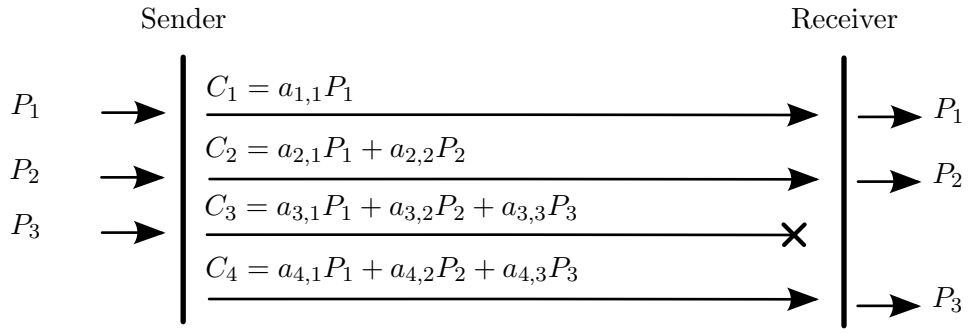



FIGURE 4 – Principe de Pipeline coding avec une taille de génération $m = 3$ et une combinaison redondante

### 2.4.3 Codage à fenêtre glissante

Le codage à fenêtre glissante est une alternative au codage réseau à générations. Il permet de lisser le processus de codage et de décodage et convient parfaitement aux protocoles qui utilisent déjà une fenêtre de transmission, par exemple TCP (comme dans TCP/NC), mais il présente l'inconvénient d'exiger une forme de retour.

Le principe est d'utiliser une fenêtre coulissante pour coder des paquets ensemble. La fenêtre avance en supprimant les anciens paquets qui ne doivent plus être inclus dans les combinaisons suivantes et en ajoutant de nouveaux paquets générés par la couche supérieure, ce qui nécessite un retour d'information depuis le processus de décodage. Les paquets qui peuvent être éliminés du processus de codage sont les paquets qui ne sont plus nécessaires pour compléter le processus de décodage. Plus précisément, ce sont les paquets qui peuvent être exprimés uniquement avec des paquets plus récents, appelés paquets vus, ou *seen* (figure 5). Les paquets vus ne sont pas nécessairement décodables, mais comme ils peuvent être exprimés sous forme de combinaisons de paquets futurs uniquement, ils seront décodables à l'avenir, même si ils ne font plus partie des combinaisons entrantes.

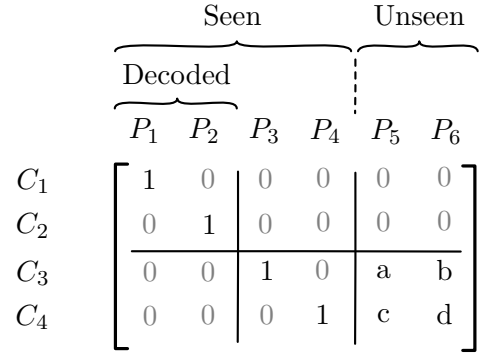



FIGURE 5 – Décodage partiel mettant en évidence les paquets vus (*seen*)

# 3 Adaptation de redondance

Un des avantages principaux du codage intra-flux est de prévenir les pertes. En utilisant celui-ci, les paquets originaux peuvent être décodés dès qu'un nombre suffisant de combinaisons ont été reçues, et ce quelque soient les combinaisons reçues. Il fonctionne à la manière d'un code à effacement (*erasure code*) d'un nœud à l'autre et permet de récupérer l'ensemble des paquets originaux en présence de pertes sur le trajet.

Il est aussi possible de tirer profit de cette propriété pour améliorer la fiabilité dans des topologies où de multiples chemins potentiels sont disponibles, par exemple les réseaux de type maillage sans fil, car les flux de bout-en bout peuvent alors transiter de manière transparente d'un chemin à l'autre selon le fonctionnement des liens.

Quand le codage réseau est déployé pour un flux unicast, il est nécessaire de deviner le nombre de combinaisons à générer au niveau de la source et des relais, afin que la destination soit en mesure de décoder les paquets.

Si la redondance est trop faible, elle peut devenir insuffisante pour compenser les pertes. Dans ce cas, le flux peut en pratique montrer des performances inférieures, à cause de groupes entiers de paquets consécutifs qui ni peuvent pas être décodés. À l'inverse, si la redondance est trop élevée, le surcoût en termes de charge pour le réseau augmente sans aucun bénéfice pour les performances d'un point de vue applicatif.

Nous pensons que la redondance est en fait un compromis entre la charge du réseau et la tolérance des applications. Selon l'application, par exemple le transfert de fichiers ou la diffusion vidéo, la tolérance aux pertes est très différente. Il est donc nécessaire de régler le facteur de redondance pour le faire correspondre aux caractéristiques du réseau. De plus, il est nécessaire d'adapter dynamiquement ce facteur si les caractéristiques du réseau changent avec le temps.

Cependant, même si des informations relatives à la qualité des liens est disponible, adapter le facteur de redondance en conséquence n'est pas une opération triviale, et plusieurs travaux reposent sur des formules empiriques menant potentiellement à des situations de sur-redondance. Nous proposons donc un modèle de redondance pour facilement évaluer une borne de redondance acceptable.

## 3.1 Estimation de la redundance

Le premier objectif de notre travail est de trouver un moyen approprié de calculer le facteur de redondance minimal à appliquer compte tenu du taux de perte du réseau, et ce de manière plus précise que la redondance moyennes. Le paramètre de redondance doit offrir une garantie correcte en termes de perte maximale tolérée au niveau de l'application.

Pour cela, la redondance doit non seulement tenir compte des caractéristiques du réseau, mais également des exigences de l'application. Lorsque les exigences de l'application sont disponibles, le facteur de redondance doit être modifié pour être suffisant pour celle-ci, mais pas plus élevé.

### 3.1.1 Borne inférieure pour la redondance

Nous nous intéressons tout d'abord à l'obtention d'un facteur de redondance minimale pour envoyer du trafic sur un lien à perte en utilisant le codage réseau. Nous voulons que cette redondance tienne compte à la fois de la perte du lien et des exigences de l'application.

Nous considérons un nœud en train d'envoyer des paquets à un autre nœud à travers un lien caractérisé par un taux de perte uniforme $p$. L'expéditeur, pour envoyer une génération de $m$ paquets, applique le taux de redondance $r$ et transmet donc $n = \lceil rm \rceil$ combinaisons.

Soit $\tau$ le taux de perte maximum toléré par l'application. On obtient la borne inférieure suivante pour $r$, qui peut être calculée à partir de $\tau$, de la taille de génération $m$, de la probabilité de perte $p$ :

$$r \geq r_{\text{bound}}(m, p, \tau) = \frac{1}{1-p}\left(\frac{1 + (1 + \sqrt{2C})^2}{2}\right) \simeq \frac{1}{1-p}\left(1 + \sqrt{2C}\right)$$

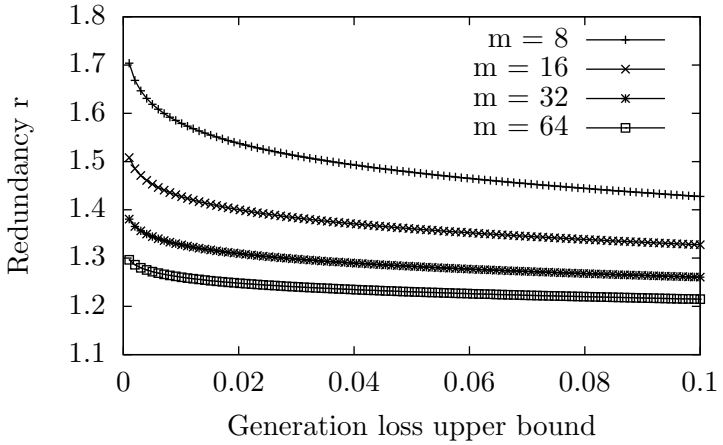



FIGURE 6 – Facteur de redondance minimal $r_{\text{bound}}$ étant donné le taux de perte toléré $\tau$ pour différentes valeurs de la taille de génération $m$ ($p = 0.1$)

### 3.1.2 Contrôle adaptatif distribué de la redondance

Maintenant que nous avons défini une redondance liée à un lien, nous voulons dériver un algorithme distribué. Le déploiement d'un algorithme centralisé pour l'adaptation de la redondance dans un réseau entier est plus simple par certains aspects mais empêche la mise à l'échelle et le déploiement pratique.

Dans cette section, nous cherchons à définir un système de codage réseau intra-flux distribué avec un contrôle de redondance adaptatif pour une transmission de données fiable.

Le système ne nécessite pas de signalisation et repose sur un routage opportuniste : chaque nœud peut entendre les paquets codés envoyés par ses voisins et peut donc participer à la transmission des paquets codés vers la destination. La redondance liée définie précédemment permet d'optimiser le nombre de combinaisons redondantes qu'un noeud doit générer vers ses voisins.

Plus précisément, lorsqu'un nœud $i$ entend une combinaison envoyée par un nœud $k$ à la destination finale $j$ (figure 7), il exécute les étapes suivantes :

— Le nœud $i$ décide s'il doit transférer le trafic. Pour ce faire, il compare sa distance à $j$ à la distance de $k$ à $j$. Si $k$ est plus proche, le paquet est supprimé, sinon le paquet est conservé en mémoire avec les paquets déjà reçus du même flux, puis on passe à la seconde étape.

— Le nœud $i$ calcule la redondance nécessaire pour ce flux en utilisant les qualités de transmission de lien de ses voisins et l'estimation de redondance présentée ci-dessus. Enfin, $i$ génère des paquets codés selon $r$ et les envoie.

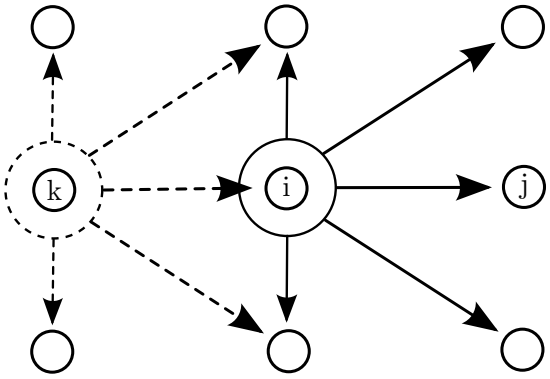



FIGURE 7 – Le nœud $i$ entend une combinaison de $k$ et envoie une nouvelle combinaison.

### 3.1.3 Résultats de la simulation

Nous évaluons ensuite notre modèle grâce à des simulations. À cette fin, nous avons développé un programme en C++ qui simule l'algorithme proposé. Le programme simule les transmissions par paquets, la perte de liens, ainsi que les opérations de codage et de décodage.

Nous évaluons l'algorithme d'estimation de la redondance dans trois topologies différentes avec une, deux et trois lignes de relais parallèles (figure 8). Les transmissions entre relais présentent une perte de lien uniforme $p$ pour simuler les interférences dans le réseau.

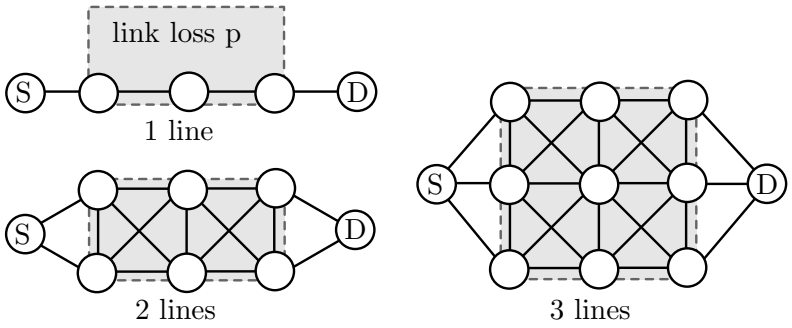



FIGURE 8 – Topologies avec différents nombres de lignes de relais avec perte de lien $p$ entre relais

Deux paramètres sont mesurés : la perte à destination et l'overhead relatif calculé comme suit :

$$\text{relative overhead} = \frac{\#\text{ paquets envoyés}}{\#\text{ nodes} \times \#\text{ paquets originaux}}$$

Nous transmettons des données depuis la source vers la destination en utilisant la taille de génération $m = 32$ avec notre algorithme d'adaptation de redondance ($\tau = 0.01$) et les autres mécanismes suivants :

— *Average*, où chaque nœud envoie avec la redondance $\frac{1}{1-p}$
— *Static*, où chaque nœud envoie avec un facteur de redondance constant prédéfini $r$

— *CodeMP-like*, où chaque nœud envoie avec la redondance $\frac{1}{1-p} + K$. La constante $K$ est prédéfinie et doit être suffisamment élevée pour que ce schéma fonctionne, mais une valeur trop élevée entraîne des situations de sur-redondance.

Avec une seule ligne de relais, seul notre algorithme réalise une perte proche de zéro à la destination (figure 9), mais il résulte en plus d'overhead que les autres (figure 10). Comme prévu, la redondance moyenne se comporte mal même avec de faibles pertes, contrairement aux autres systèmes utilisant une redondance statique supplémentaire.

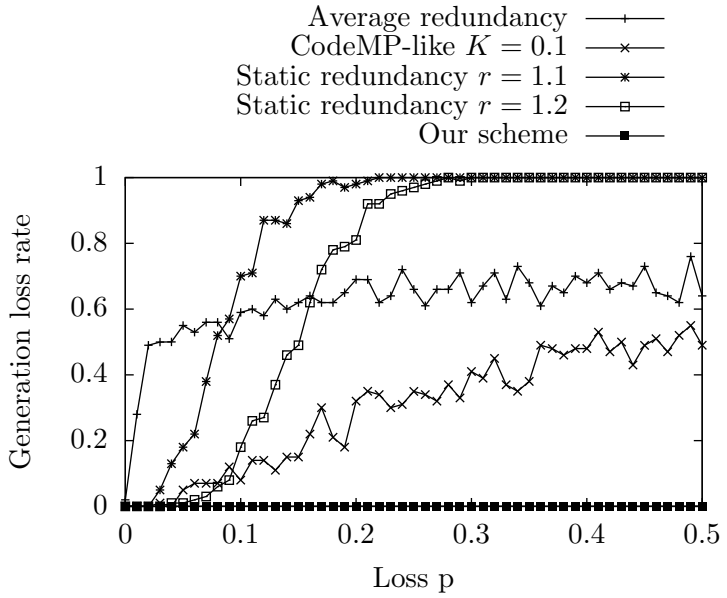



FIGURE 9 – Perte de génération à destination selon la perte $p$ avec une ligne de nœud de relais

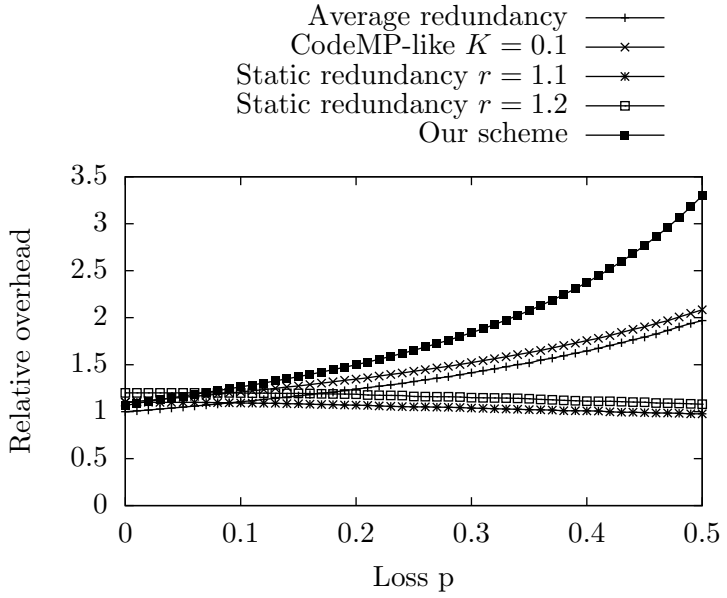



FIGURE 10 – Overhead relatif selon la perte $p$ avec une ligne de nœud de relais

Avec plus d'une ligne de relais, le relais opportuniste est suffisants pour que les différents mécanismes atteignent une perte presque nulle à destination. Cependant, l'overhead est beau-

coup plus bas avec notre algorithme, puisqu'il n'innonde pas le réseau (figure 11). Ceci est encore plus visible avec trois lignes de relais (figure 12).

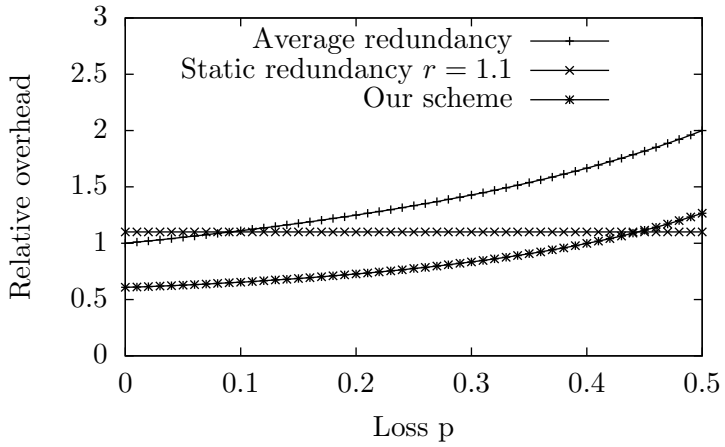



FIGURE 11 – Overhead relatif selon la perte $p$ avec deux lignes de nœud de relais

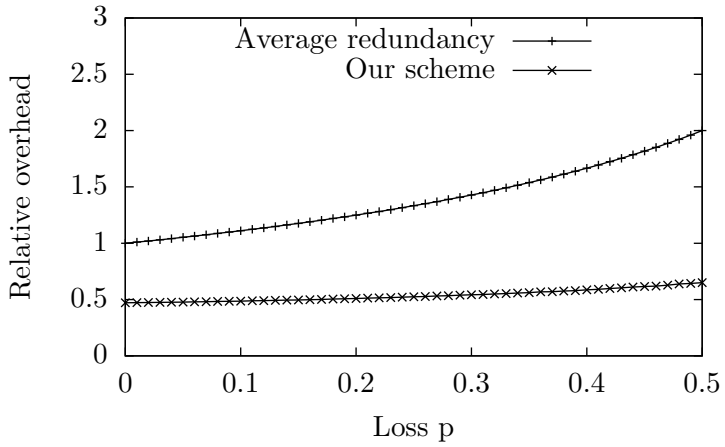



FIGURE 12 – *Overhead* relatif selon la perte $p$ avec trois lignes de nœud de relais

## 3.2 Prise en compte des contraintes

Dans les réseaux sans fil maillés, les nœuds ont souvent de fortes contraintes en matière de consommation d'énergie et de puissance de traitement. Dans ce cas, le codage et la transmission peuvent représenter des opérations coûteuses. Une façon de réduire le coût du codage consiste à distribuer les opérations de codage sur un nombre limité de noeuds tout en essayant d'optimiser le gains en termes de performances.

Pour répondre à ce défi, notre objectif est de proposer un algorithme distribué qui prend en compte la capacité de chaque nœud de réseau à contribuer aux opérations de codage. Chaque nœud définit sa contribution relative en fonction de ses propres contraintes. Celle-ci est prise en compte lors de la transmission des combinaisons de paquets via les noeuds du réseau, tout en garantissant le décodage à destination.

### 3.2.1 General principle

Chaque nœud $i$ peut définir un facteur de contribution $\alpha_i \in ]0, 1]$ qui indique sa contribution relative à la transmission par le réseau. Les nœuds définissent leur facteur en fonction de leurs propres contraintes, et chaque nœud diffuse son facteur et celui de chacun de ses voisins. Les combinaisons sont transmises dans le réseau avec la contrainte de décodage à destination, tout en gardant la contribution de chaque nœud par rapport à ses voisins proportionnelle à son facteur $\alpha_i$.

Par exemple, un nœud avec un facteur $\alpha$ dix fois plus élevé que ses voisins générerera et enverra en moyenne dix fois plus de combinaisons. Ce mécanisme permet de faire en sorte que les combinaisons soient générées en priorité par des nœuds possédant les capacités de le faire (figure 13).

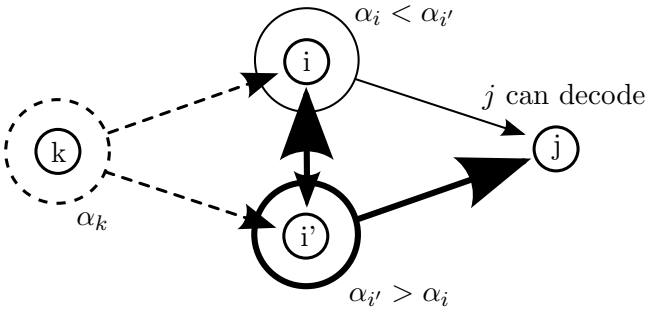



FIGURE 13 – Nodes $i$ et $i'$ relaient le traffic avec différentes contributions.

### 3.2.2 Résultats de simulation

Le facteur de contribution $\alpha$ permet d'équilibrer le trafic entre un nœud et ses voisins. Nous montrons son effet dans une topologie avec plusieurs lignes (figure 14) en réduisant $\alpha$ sur les nœuds dans la première ligne.

Lorsque plusieurs nœuds peuvent retransmettre les paquets du même flux, leur contribution relative respective sera proportionnelle à leur facteur $\alpha$. Par exemple, un nœud avec $\alpha = 0.5$ aura tendance à générer la moitié du nombre de combinaisons d'un nœud avec $\alpha = 1$, ce qui lui permet d'économiser de l'énergie si nécessaire.

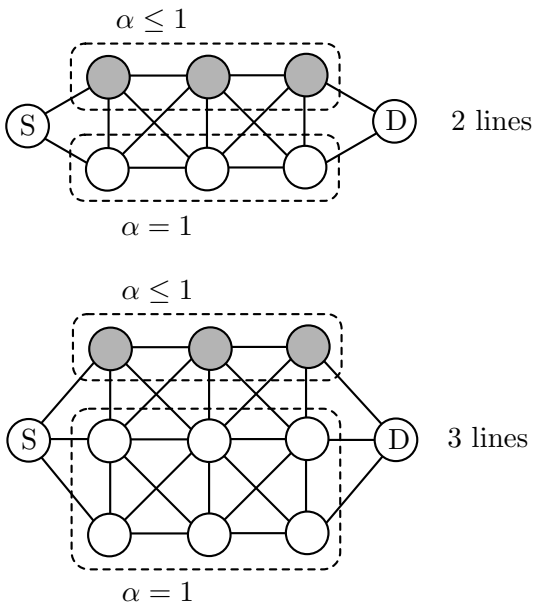



FIGURE 14 – Topologies simulées avec 2 ou 3 lignes

Comme prévu, abaisser le facteur $\alpha$ sur la première ligne a pour effet de détourner le trafic vers les autres lignes où les nœuds ont $\alpha$ égal à 1, en particulier à la seconde (figure 15).

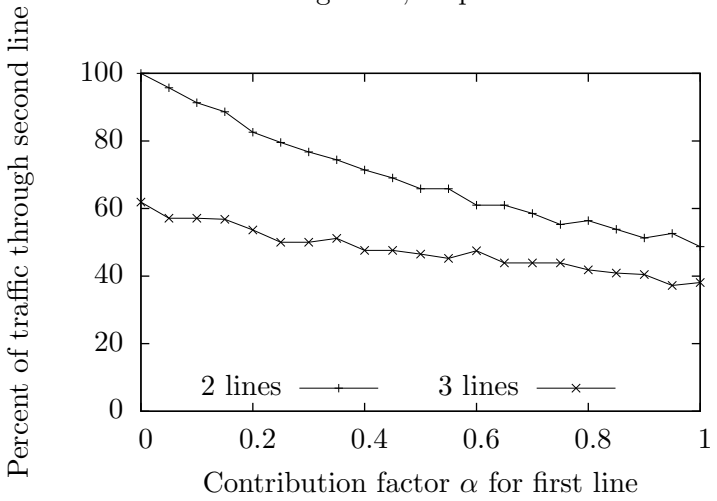



FIGURE 15 – Effet du facteur $\alpha$ sur la première ligne

# 4 Interaction entre le codage réseau et TCP

Plusieurs auteurs supposent que l'obstacle majeur au déploiement du codage réseau est que la manière d'intégrer naturellement cette technique aux réseaux existants n'est pour l'instant pas évidente. En particulier, un déploiement incrémental est difficile et il est difficile du prédire comment le codage réseau va se comporter une fois déployé en situation réelle.

L'interaction du codage réseau avec le protocole TCP est l'un des problèmes majeurs, puisque TCP est aujourd'hui le protocole de transport prédominant sur Internet.

Il est bien documenté que TCP fonctionne mal au dessus de liens présentant des pertes de paquets, parce qu'il interprète les pertes comme des signaux de congestion. En effet, les pertes de paquets sont extrêmement peu communes sur les liens filiaires, c'est pourquoi traditionnellement, la seule cause de pertes de paquets est la congestion du réseau entraînant des pertes au niveau d'un nœud.

Comme le codage réseau intra-flux est capable d'améliorer la fiabilité et la performance des flux, il semble tout à fait adapté pour corriger le fonctionnement de TCP. En effet, il permet de cacher les pertes de lien, rendant inutiles les retransmissions de segments et les réductions inutiles de la fenêtre de congestion de TCP.

Cependant, le déploiement du codage réseau conjointement à TCP pose un problème vis-à-vis de l'équité entre flux. En conditions normales, les flux TCP partagent la capacité disponible grâce à leur algorithme de contrôle de congestion de type augmentation additive/retrait multiplicatif (AIMD). Cependant, la plupart du temps, le signal de congestion est la perte de paquets. Le codage réseau intra-flux cache les pertes aux flux TCP indépendemment de leur cause, les pertes de congestion sont donc cachées comme le sont les pertes de lien. Pour cette raison, les flux TCP fonctionnant au-dessus du codage réseau pourraient prendre une part inéquitablement important de la capacité disponible lorsqu'il sont en concurrence avec des flux non codés. Dans les pires cas, il se pourrait que les flux non codés s'arrêtent de transmettre.

Étant donné que TCP est de loin le protocole de transport le plus utilisé sur Internet, plusieurs travaux se sont naturellement concentrés sur son interaction avec le codage réseau. Par exemple, TCP/NC, qui introduit une couche de codage réseau entre IP et TCP a été testé avec deux flux codés concurrents du même type, mais le problème d'équité demeure en cas de concurrence avec des flux non codés.

Dans ce travail, nous discutons de ce problème d'équité entre flux codés et flux non codés et nous définissons une nouvelle mesure d'équité pour déterminer les implications du fonctionnement de flux codés en parallèle de flux non codés.

## 4.1 Sensibilité aux pertes et inéquité

Lors de l'exécution de TCP sur le Pipeline coding, la couche de codage agit comme une couche de *Forward Error Correction* (FEC) transparente (figure 16), dissimulant efficacement les pertes de la couche TCP.

### 4.1.1 Pertes de liens et pertes de congestion

Étant donné que les pertes sont indistinctement cachées par TCP par Pipeline coding, il est probable que le mécanisme de réduction de fenêtre TCP ne soit pas déclenché comme il devrait.

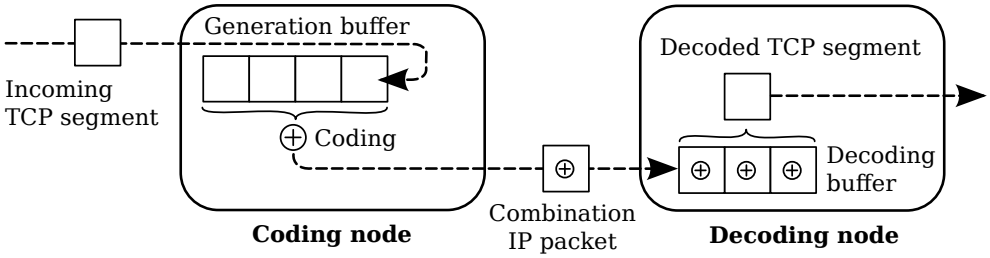



FIGURE 16 – TCP sur Pipeline coding

De plus, cette situation s'aggrave lorsque le facteur de redondance du code augmente.

Pour vérifier cela, après avoir configuré une topologie de goulot d'étranglement et aucune perte de lien dans le simulateur réseau *ns-3*, nous surveillons les paquets perdus du fait du débordement de la file d'attente au niveau du routeur en amont du goulot d'étranglement, qui correspondent aux pertes de congestion.

Ensuite, nous comparons l'évolution de la fenêtre entre TCP NewReno sans codage et TCP NewReno avec codage dans une situation extrême, avec la taille de génération $g = 64$ et le taux de redondance $R = 1,25$. Dans notre implémentation, tous les noeuds intermédiaires suppriment les paquets redondants et recodent les innovants avec la redondance $R$, de sorte que sur chaque lien, la redondance du code réel est de $R$, indépendamment du taux de perte sur les liens précédents.

Sur la figure 17 sont représentées les évolutions des fenêtres de congestion, et les barres verticales marquent les heures où un paquet est supprimé en raison du débordement de la file d'attente, c'est-à-dire lorsque survient une perte de congestion. Le comportement observable des mécanismes de contrôle de la congestion TCP est radicalement différent entre le cas codé et le cas non codé.

Alors que TCP sans codage nécessite seulement une perte de congestion pour détecter la congestion (en recevant des ACK dupliqués) et pour réagir en divisant sa fenêtre, il faut que des dizaines d'entre eux réagissent par rapport au codage, car la détection de congestion nécessite de perdre une génération, et cela ne peut se produire que lorsque plus de $(r-1)g = 16$ paquets sont perdus. Pour cette raison, les flux surcodés sont plus lents à réagir à la congestion que ceux non codés.

L'un des problèmes est qu'un peu de surcodage est nécessaire, et la redondance ne doit pas être définie aussi basse que $R_{\min} = \frac{1}{1-p}$ (où $p$ est le taux moyen de perte mesurée), et ce afin de compenser les pertes en moyenne, une redondance supplémentaire est nécessaire pour éviter de perdre des générations qui rencontrent plus de pertes statistiquement ou de réduire légèrement la qualité des liaisons.

### 4.1.2 Cas de flux concurrents

Si on considère maintenant deux flux TCP concurrents dans ce même goulot d'étranglement, le premier codé et le second non codé, deux facteurs différents empêchent le flux non codé d'utiliser toute sa part de débit disponible :

— Les pertes de lien sont cachées au flux codé, mais pas à celui non codé. Par conséquent, le contrôle de congestion du flux non codé les interprète comme des pertes de congestion,

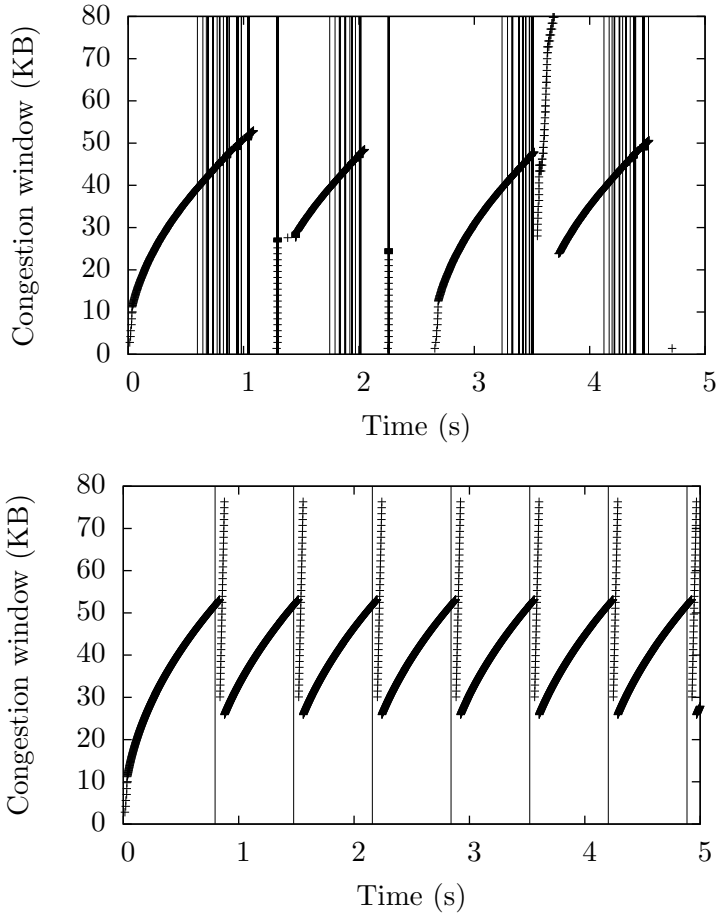



FIGURE 17 – Fenêtre de congestion avec (haut) et sans codage (en bas)

par conséquent, l'expéditeur TCP ajuste sa taille de fenêtre comme si le réseau était congestionné. Le flux codé ne souffre pas de ces pertes, de sorte qu'il peut prendre une plus grande partie de la capacité disponible.

— Certaines pertes de congestion sont cachées au flux codé, mais aucune n'est cachée au flux non codé, de sorte que ce dernier est plus sensible à la congestion réelle dans le réseau et réagit plus rapidement, laissant une plus grande partie de la capacité au flux codé.

Le premier mécanisme est souhaitable, car c'est un simple résultat du fait que TCP fonctionne mieux avec le codage, mais le second est clairement indésirable. Nous voulons mesurer quelle partie de la meilleure performance d'un flux codé provient de sa sensibilité réduite aux pertes de lien et quelle partie provient d'une sensibilité plus élevée des flux concurrents à la congestion.

Dans nos résultats (figure 18), nous constatons une augmentation de la différence de débit entre les flux à mesure que la redondance $r$ augmente, le débit du flux codé codé prenant graduellement la plus grande partie de la capacité disponible. Cependant, il semble y avoir une limite et le flux codé ne prend jamais toute la capacité, même avec des taux de redondance relativement élevés comme $r = 1,5$. Cela signifie que la corrélation des pertes de congestion est suffisante pour que le flux codé réagisse à un moment donné.

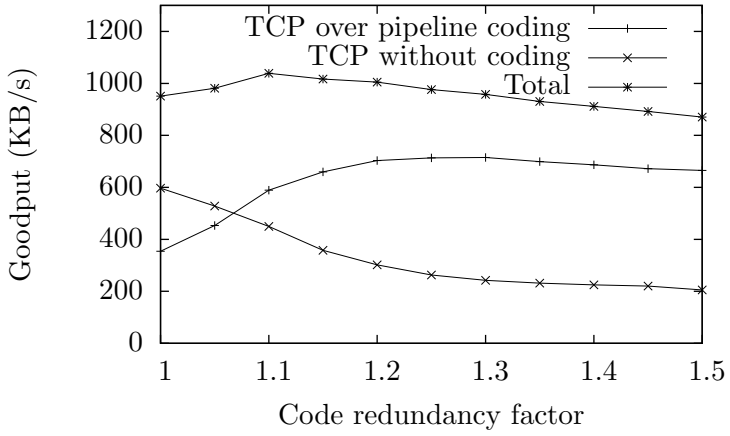



FIGURE 18 – Comparaison de débit entre des flux concurrents, un avec codage à différents facteurs de redondance et un sans ($p = 1\%$)

### 4.1.3 Mesurer l'équité

Notre objectif est d'obtenir une mesure qui est sensible au débit que les flux codés prennent injustement au détriment des codes non codés, mais pas sensible au fait que les flux codés fonctionnent mieux indépendamment des flux non codé.

Soit une allocation de débit pour $n$ flux concurrents pour un capacité $c$ sur un chemin à perte $(x_0, ..., x_n)$. Les premiers $k$ flux sont codés et les autres $n - k$ ne sont pas codés ($k > 0$). Soit $x'$ le débit moyen qu'obtiendrait un flux dans la même allocation si les flux codés n'étaient pas codés. Alors, on définit $\lambda$ comme le ratio entre le débit, dans une allocation équitable, d'un flux non codé (égal à $x'$) et celui d'un flux non codé :

$$\lambda = \frac{kx'}{c - (n-k)x'}$$

Nous introduisons un nouvel indice d'équité $J'$, dérivé de l'indice de Jain et prenant en compte cette correction :

$$J'(x_1, ..., x_n) = \frac{(\lambda \sum_{i=1}^{k} x_i + \sum_{i=k+1}^{n} x_i)^2}{N(\lambda^2 \sum_{i=1}^{k} x_i^2 + \sum_{i=k+1}^{n} x_i^2)}$$

Cet indice d'équité modifié a les mêmes propriétés que l'indice de Jain, mais le meilleur cas correspond à une répartition équitable telle que voulue dans notre cas et non à une allocation uniforme. Il note l'équité avec une valeur comprise entre $\frac{1}{n}$ (pire des cas) et 1 (le meilleur des cas, l'allocation est équitable).

En particulier si $n = 2$ et $k = 1$ :

$$\lambda = \frac{x'}{c - x'}$$

$$J'(x_1, x_2) = \frac{(\lambda x_1 + x_2)^2}{2(\lambda x_1)^2 + 2x_2^2}$$

La figure 19 souligne que l'allocation est en réalité plus juste que ce que l'on pouvait croire, car le flux codé prend une partie de la ressource dont le flux non codé ne pouvait pas prendre

de toute façon. Lorsque la redondance $r$ s'approche de valeurs plus élevées, l'équité ne baisse pas et reste relativement élevée autour de $0,85$. Cela indique même avec trop de redondance, les flux TCP codés n'écrasent pas les flux non codés. Cela signifie que les pertes de congestion sont assez corrélées pour déclencher une perte de génération et donc le contrôle de congestion de TCP.

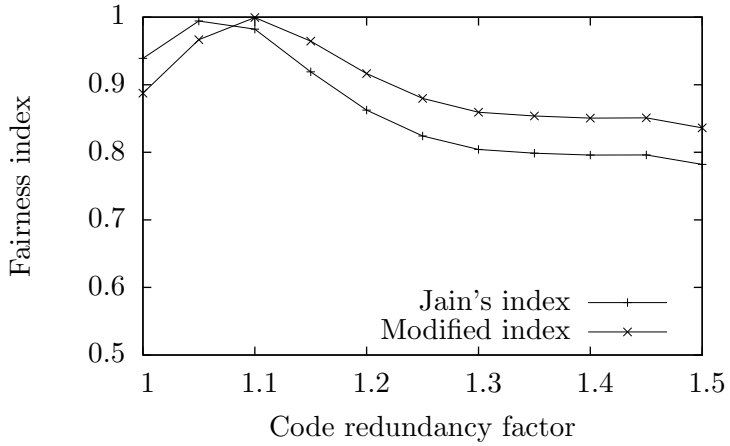



FIGURE 19 – Comparaison des indices à différents facteurs de redondance pour un flux codé et un non codé ($p = 1\%$)

## 4.2 Multi-Path TCP sur codage réseau

Multi-path TCP (MPTCP) est une extension TCP récente standardisée dans RFC 6824 par l'IETF. Il permet de répartir une unique connexion TCP sur plusieurs chemins avec des adresses différentes, mais il cache la complexité à la couche supérieure en gardant l'API de socket TCP standard. Il vise à optimiser l'utilisation des ressources réseau et à offrir un meilleur débit et une meilleure robustesse.

MPTCP a d'abord été conçu pour fournir de meilleurs performances dans des centres de données. En règle générale, MPTCP cible les hôtes multihomés dans les centres de données afin d'augmenter la bande passante inter-datacenter.

Récemment, son intérêt a été démontré pour les appareils mobiles dotés de radios multiples. Par exemple, les smartphones connectés à la fois WiFi et LTE peuvent en profiter pour augmenter les débits, améliorer la commutation entre WiFi et LTE, et fournir des performances stables malgré des dégradations ou des pannes intermittentes sur l'un ou l'autre canal.

Du point de vue du réseau, MPTCP utilise des sous-flux entièrement compatibles TCP pour différentes paires d'adresses IP, sur lesquelles le flux de données original est mappé. Cette rétro-compatibilité complète permet de déployer MPTCP sans problèmes de traversée de *middlebox*, appareils qui interprètent le trafic TCP.

De la même manière que TCP est sensible aux pertes de lien, MPTCP souffre de dégradations de performances lorsqu'il est utilisé sur des chemins subissant des pertes aléatoires. Non seulement les paquets perdus doivent être retransmis, mais l'algorithme de contrôle de congestion tend à mal interpréter les pertes de liens car les signaux de congestion, conduisant à une fenêtre de congestion inutilement réduite et à un débit trop faible.

Dans cette partie, nous étudions l'impact du Pipeline coding sur les performances de MPTCP. Nous avons mis en œuvre une expérience en utilisant l'implémentation MPTCP de référence de l'Université Catholique de Louvain, et en ajoutant une couche du codage réseau au niveau IP. Nous croyons que l'utilisation du codage réseau au niveau IP est une approche pertinente puisqu'elle fournit une rétro-compatibilité TCP et évite les problèmes de traversée de *middle-box*. La couche du codage réseau permet de compenser les pertes de liens empêchant à la fois la réduction des fenêtres de congestion et les problèmes de blocage de ligne de ligne au planificateur. Nous avons effectué une évaluation de performance de MPTCP codé sur un réseau émulé relativement réaliste. Une comparaison des performances MPTCP avec et sans codage est fournie.

Le principe est de garder la pile MPTCP non modifiée et de faire fonctionner chaque sous-flux sur une session de Pipeline coding (figure 20).

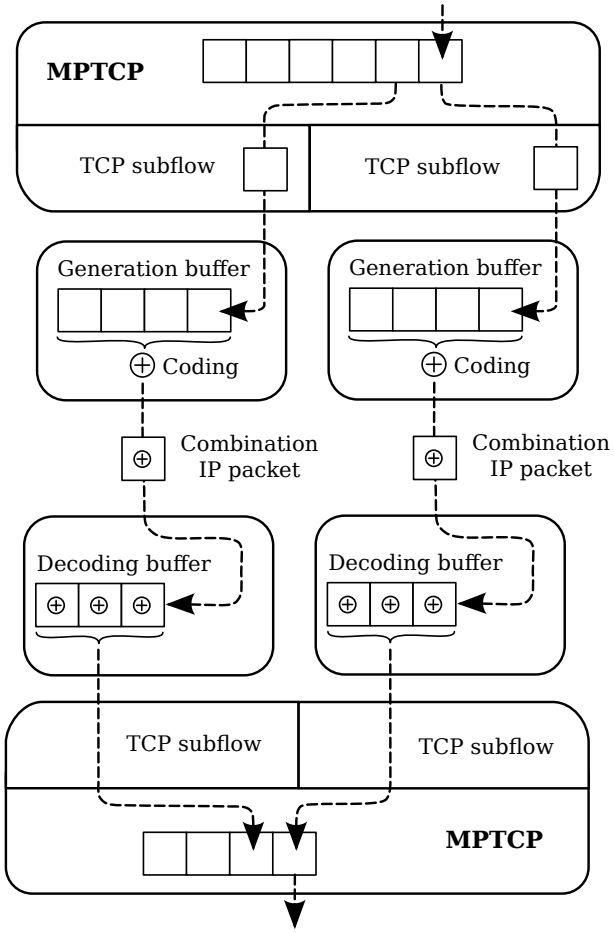



FIGURE 20 – MPTCP avec deux sous-flux sur le Pipeline coding intra-flux

Du côté du récepteur, les paquets de chaque sous flux sont décodés indépendamment. Lorsqu'un paquet peut être décodé, il passe immédiatement à MPTCP. Le taux de redondance, *c'est-à-dire*, le nombre de combinaisons pour chaque paquet d'origine, est réglable, ce qui permettra d'observer son impact sur les performances.

Le codage réseau devrait avoir un impact positif sur MPTCP lors de l'exécution sur des liens à perte, car il protège non seulement les sous-flux TCP des pertes de liens, mais aussi empêche les problèmes de planification lorsqu'ils réagissent aux pertes sur les sous-flux en retransmettant

sur un autre lien.

### 4.2.1 Configuration réseau émulée

Nous avons mené des tests sur l'implémentation de référence MPTCP pour le noyau Linux développé par l'Université Catholique de Louvain, version 0.90. L'émulation de réseau est réalisée avec le simulateur ns-3 fonctionnant en mode temps réel. Deux machines virtuelles UML (*User-Mode Linux*), une pour le client et une pour le serveur, sont installées sur l'hôte parallèlement au simulateur. Le trafic à partir des interfaces des deux machines virtuelles est acheminé vers les nœuds correspondants dans ns-3. Lorsqu'un transfert de fichier est exécuté entre les machines, les sous-flux sont ouverts sur le réseau émulé (figure 21).

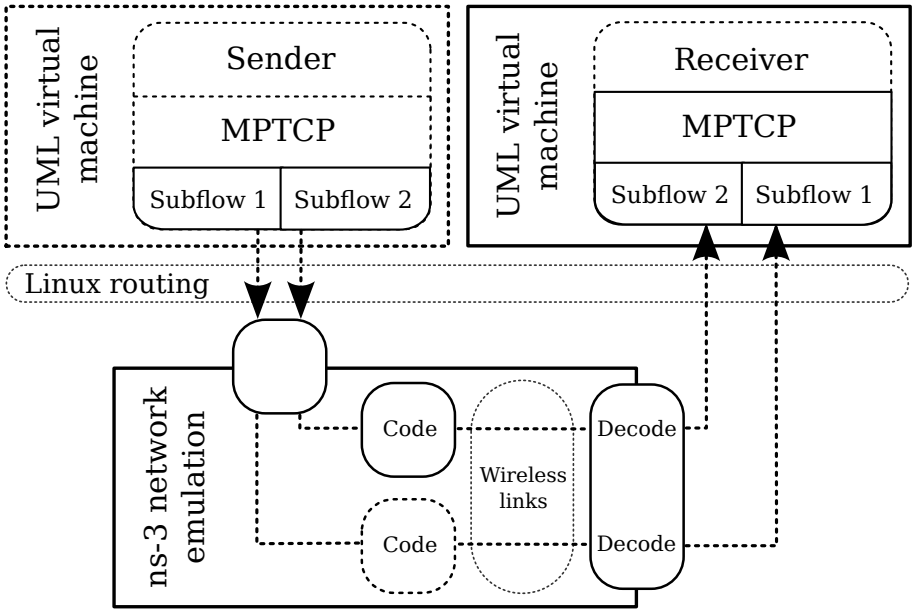



FIGURE 21 – Configuration d'émulation de réseau avec ns-3 et deux machines virtuelles *User-Mode Linux*

Le réseau émulé se compose de quatre nœuds : deux nœuds comme points d'accès sans fil, un pour le serveur et un pour le client avec deux interfaces. Les nœuds clients et serveurs utilisent la fonction pont ns-3 pour rendre leurs interfaces disponibles à partir de l'hôte Linux et le trafic est acheminé vers et à partir des machines virtuelles correspondantes avec des tables de routage spécifiques sur l'hôte. Les liens sans fil indépendants sont imités entre le nœud client et les points d'accès (figure 22).

La figure 23 montre le bénéfice du codage réseau pour le débit de MPTCP. Au fur et à mesure que le taux de perte augmente, le débit applicatif sans codage s'effondre alors que celui avec le codage est protégé, offrant un gain constant jusqu'à des taux de perte élevés.

La figure 24 compare le débit applicatif pour différents facteurs de redondance $r$. L'augmentation de la redondance augmente le taux de perte maximal à partir duquel le débit s'effondre, mais augmente aussi l'overhead.

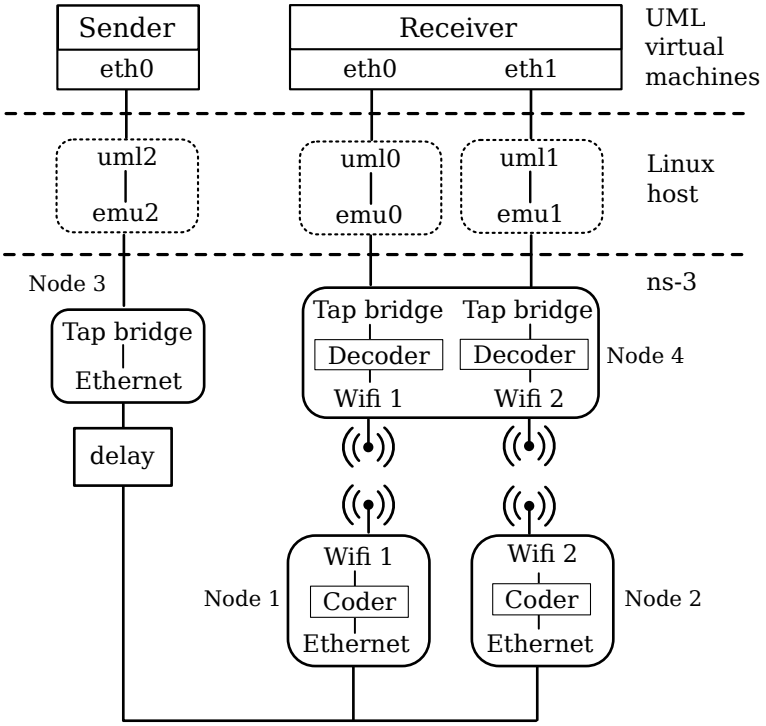



FIGURE 22 – Détails internes de l'émulation avec ns-3

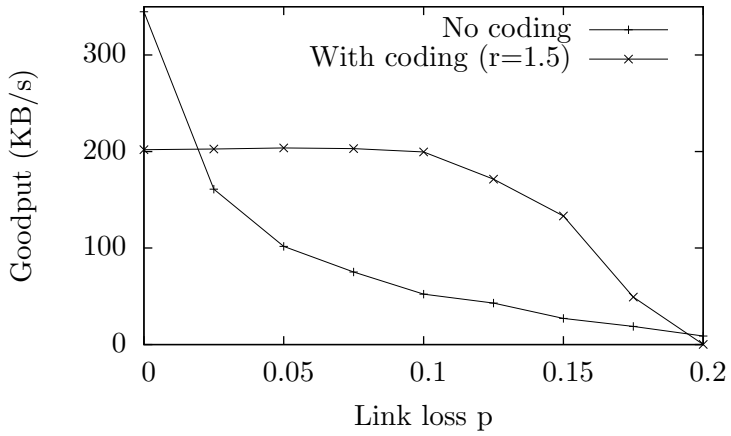



FIGURE 23 – Débit applicatif pour MPTCP avec et sans Pipeline coding sur deux liens selon le taux de perte $p$

### 4.3 Intégration du codage réseau dans MPTCP

Dans cette dernière partie, nous proposons une méthode alternative pour intégrer le codage réseau dans MPTCP, un protocole pour envoyer des segments codés au dessus des sous-flux TCP. Cette approche permet potentiellement de pallier les phénomènes de blocage tout en garder une retro-compatibilité complète avec TCP pour la traversée de pare-feux ou de réseaux restrictifs.

En effet, MPTCP utilise une fenêtre globale mappée aux fenêtres des sous-flux, et un problème commun rencontré avec cette approche, en particulier avec des liens sans fil peu

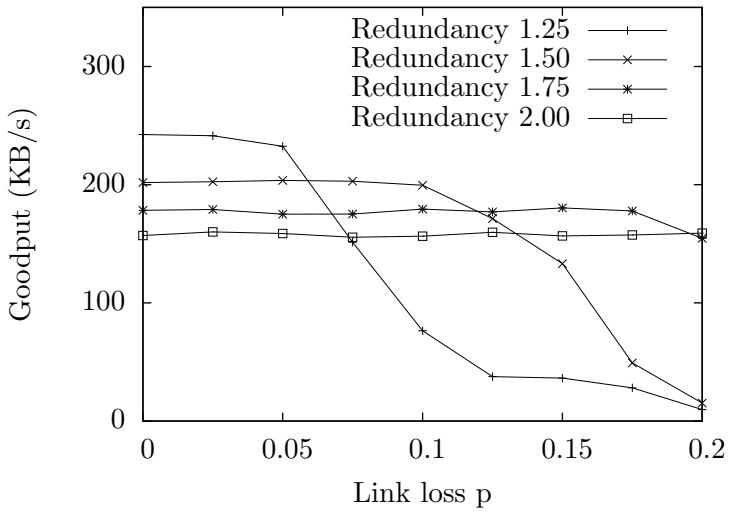



FIGURE 24 – Débit applicatif pour MPTCP avec Pipeline coding selon la redondance $r$

fiables, est le blocage de la tête de ligne (ou *head-of-line blocking*).

Il se produit lorsqu'un lien se dégrade soudainement : la fenêtre globale MPTCP ne peut pas avancer parce que les paquets programmés sur le lien défaillant sont manquants à destination (figure 25). Jusqu'à ce que ces paquets finissent par être reçus ou soient réinjectés sur des autres liens, la connexion peut être bloquée, même si d'autres sous-flux fonctionnent bien. Par exemple, ce scénario peut se produire lorsqu'un utilisateur qui télécharge un flux vidéo en WiFi et LTE à la fois s'éloigne du point d'accès WiFi. La dégradation de la qualité de la liaison WiFi en raison de la distance pourrait retarder l'intégralité du flux de données MPTCP et geler la vidéo.

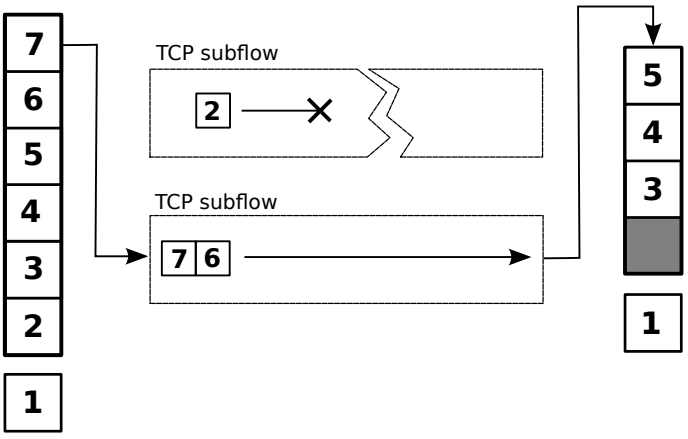



FIGURE 25 – Problème de *head-of-line blocking* rencontré avec MPTCP

Notre objectif est d'accroître la résilience de MPTCP en résolvant le problème du *head-of-line blocking*. À cette fin, nous définissons et testons une extension MPTCP pratique, MultiPath Coded TCP (MPC-TCP), qui permet à la source TCP de transmettre des combinaisons linéaires de segments plutôt que des segments de données sur des sous-flux TCP. Du côté du récepteur, les données peuvent être décodées dès que suffisamment de combinaisons sont disponibles, peu importe les sous-flux utilisés pour les transmettre (figure 26). Ce système offre trois avantages : la rétrocompatibilité MPTCP, la compatibilité TCP complète pour la traversée de réseaux

restrictifs, et la simplicité d'implémentation.

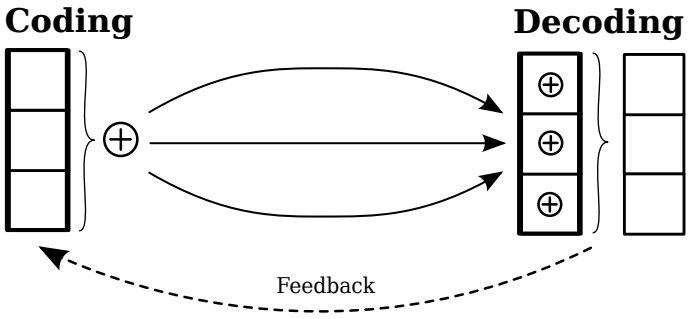



FIGURE 26 – Principe simplifié de MultiPath Coded TCP (MPC-TCP)

Nous avons développé notre propre implémentation dans le noyau Linux à partir de l'implémentation de référence MPTCP v0.90 de l'Université Catholique de Louvain.

Les évaluations ont été effectuées avec un réseau émulé dans ns-3 pour mesurer la capacité du codage réseau pour résoudre le problème de blocage. Dans ce chapitre, nous nous concentrons uniquement sur la résolution des problèmes de blocage de la tête de ligne et nous avons conservé des sous-flux entièrement compatibles TCP, préférant une rétrocompatibilité sur les performances maximales sur les liens à perte de poids.

Dans la plupart des autres implémentations, les contraintes techniques ne sont que peu ou pas discutées, ici, à l'inverse, la mise en œuvre pratique et la compatibilité sont nos principales préoccupations.

# 5 Conclusion et résumé des contributions

Dans ce travail, nous nous sommes concentrés tout particulièrement sur l'application du codage intra-flux aux flux unicasts sur des réseaux sans fil. Le principal objectif est d'améliorer la fiabilité des transferts de données sur des liens sans fil, et de discuter des opportunités de deploiement et des performances.

Dans une première partie de ce travail, nous avons dérivé une borne minimale pour la redondance du codage pour estimer celle-ci selon la qualité de lien et le taux maximum de pertes tolérable par l'application. Nous proposons ensuite un algorithme distribué dans un réseau collaboratif pour adapter la redondance et permettre un réception fiable des données. L'algorithme permet à l'émetteur d'utiliser de manière opportuniste plusieurs chemins à travers le réseau pour router les paquets jusqu'à la destination tout en offrant un contrôle de redondance optimisé et une réception suffisamment fiable.

Comme les opérations du codage réseau sont coûteuses en termes de calcul, de stockage et d'énergie, nous avons ensuite présenté une amélioration de l'algorithme précédent qui considère la capacité de chaque nœud à contribuer au processus de codage. Les contraintes sont prises en compte pendant la transmission des combinaisons à travers le réseau tout en garantissant le décodage à destination.

Dans une seconde partie, nous avons étudié l'interaction du codage réseau avec TCP et son extension MPTCP. L'objectif de notre étude a tout d'abord été l'impact des problèmes d'équité qui se posent lorsque des flux TCP non codés partagent un goulet d'étranglement avec des flux TCP codés. Pour évaluer correctement l'équité dans ce cas spécifique, nous avons introduit un nouvel indice d'équité.

Ensuite, nous avons exploré deux approches différentes pour améliorer les performances de MPTCP avec l'introduction du codage intra-flux. Tout d'abord, MPTCP étant fondé sur les mêmes mécanismes que TCP, il est sujet aux mêmes défauts, en particulier en ce qui concerne la sensibilité aux pertes de lien. Pour cette raison, nous avons tout d'abord étudié les bénéfices de faire fonctionner TCP au dessus du codage réseau.

Dans un dernier temps, nous avons envisagé la possibilité d'utiliser le codage réseau pour résoudre le problème de blocage en tête de file de MPTCP. Pour cela, nous avons conçu et développé une extension de MPTCP appelée MultiPath Coded TCP (MPC-TCP) à partir de l'implémentation de référence pour Linux. L'idée fondamentale est d'implémenter le codage réseau directement au dessus de MPTCP plutôt qu'en dessous, afin d'augmenter la résilience de ce dernier tout en garantissant une retro-compatibilité stricte avec TCP, et permettre de franchir les envionnements réseau les plus restrictifs (notamment en présence de systèmes qui réinterprètent le trafic).

Étant donné que les implémentations du codage réseau sont encore rares et que MPTCP n'est pas encore implémenté dans les simulateurs, cette recherche a nécessité un travail important de mise en œuvre et de construction de bancs de tests logiciels. Pour l'instant, les résultats sont limités et devront donc être étendus à l'aide de l'infrastructure développée.

# Fiabilité et problèmes de déploiement du codage réseau dans les réseaux sans fil

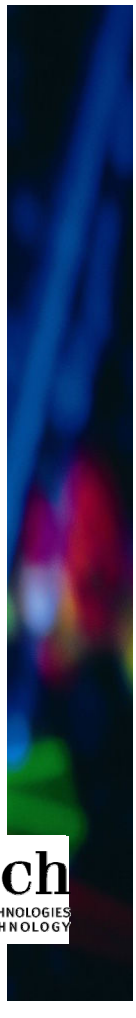
**Paul-Louis AGENEAU**

**RÉSUMÉ :** Même si les réseaux de données ont beaucoup évolué au cours des dernières décennies, les paquets sont presque toujours transmis d'un nœud à l'autre comme des blocs de données inaltérables. Cependant, ce paradigme fondamental est aujourd'hui remis en question par des techniques novatrices comme le codage réseau, qui promet des améliorations de performance et de fiabilité si les nœuds sont autorisés à mixer des paquets entre eux.
Les réseaux sans fil manquent de fiabilité en raison des obstacles ou interférences que subissent les liens sans fil, et ces problèmes peuvent empirer dans des topologies maillées avec de multiples relais potentiels. Dans ce travail, nous nous concentrons sur l'application du codage réseau intra-flux aux flux unicast dans les réseaux sans fil, avec pour objectif d'améliorer la fiabilité des transferts de données et de discuter des opportunités de déploiement et des performances.
Tout d'abord, nous proposons une borne inférieure pour la redondance, puis un algorithme opportuniste distribué, pour adapter le codage aux conditions du réseau et permettre la livraison fiable des données dans un réseau sans fil maillé, tout en prenant en compte les besoins de l'application. En outre, puisque les opérations requises pour le codage réseau sont coûteuses en termes de calcul et de mémoire, nous étendons cet algorithme pour s'adapter aux contraintes physiques de chaque nœud.
Ensuite, nous étudions les interactions du codage intra-flux avec TCP et son extension MPTCP. Le codage réseau peut en effet améliorer les performances de TCP, qui ont tendance à être plus faibles sur les liens sans fil, moins fiables. Nous observons l'impact des problèmes d'équité qui se posent quand des flux codés fonctionnent en parallèle avec des flux traditionnels non codés. Pour finir, nous explorons deux manières différentes d'améliorer les performances de MPTCP dans les environnements sans fil : le faire fonctionner sur du codage réseau, et implémenter directement le codage directement dans le protocole MPTCP tout en préservant sa compatibilité avec TCP.

**MOTS-CLEFS :** codage réseau, intra-flux, réseau sans fil, réseau maillé, routage opportuniste, redondance, fiabilité, équité, TCP, MPTCP

**ABSTRACT :** Even if packet networks have significantly evolved in the last decades, packets are still transmitted from one hop to the next as unalterable pieces of data. Yet this fundamental paradigm has recently been challenged by new techniques like network coding, which promises network performance and reliability enhancements provided nodes can mix packets together.
Wireless networks rely on various network technologies such as WiFi and LTE. They can however be unreliable due to obstacles, interferences, and these issues are worsened in wireless mesh network topologies with potential network relays. In this work, we focus on the application of intra-flow network coding to unicast flows in wireless networks. The main objective is to enhance reliability of data transfers over wireless links, and discuss deployment opportunities and performance.
First, we propose a redundancy lower bound and a distributed opportunistic algorithm, to adapt coding to network conditions and allow reliable data delivery in a wireless mesh. We believe that application requirements have also to be taken into account. Since network coding operations introduce a non negligible cost in terms of processing and memory resources, we extend the algorithm to consider the physical constraints of each node.
Then, we study the interactions of intra-flow coding with TCP and its extension MPTCP. Network coding can indeed enhance the performances of TCP, which tends to perform poorly over lossy wireless links. We investigate the pratical impact of fairness issues created when running coded TCP flows besides legacy non-coded TCP flows. Finally, we explore two different ways to enhance the performance of MPCTP in wireless environments : running it over network coding, and implementing the coding process directly in MPTCP while keeping it fully TCP-compatible.

**KEYWORDS :** network coding, intra-flow, batch coding, pipeline coding, wireless network, mesh network, opportunist routing, redundancy, reliability, fairness, TCP, MPTCP



TELECOM ParisTech

EDiTE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY