Università degli Studi di Padova

Department of Information Engineering

Master Thesis in Telecommunication Engineering

Analysis and simulation of feedback in network coded transmissions

Supervisor
Leonardo Badia
Università di Padova

Co-supervisor
Frank Wilhem
TU Dresden

Master Candidate
Alberto Piovan

Abstract

Nowadays mobile devices are provided with multiple interfaces so as to be adaptive to different use scenarios. However old-fashioned protocols do not allow using these interfaces in parallel to increase performance of the end user. Multipath protocols such as MPTCP have been proposed to fill this gap, offering increased theoretical throughput and robustness due to resources pooling, but at the same time they suffer from low performance in case of heterogeneous paths and a complex retransmission management in case of losses. This thesis aims firstly to study a new approach to this problem, introducing multipath protocols that exploit Network Coding. The particular structure of Network Coding enables to overcome the aforementioned issues simplifying the scheduling of packets among the links and the retransmission management. Secondly, we focus on the management of feedbacks and retransmissions in the presented protocols, in particular the impact that these have on the overall performance.

Specifically, we will propose two multipath transmission protocols, both based on Network Coding, and will study a feedback scheme compatible with them that allows to exploit the feature of the coding scheme. Moreover, these protocols will be implemented in a Python simulator, and a thorough simulation campaign will be conducted in order to evaluate the performance especially for what concerns overhead and feedbacks.

Sommario

I dispositivi mobili dispongono oggigiorno di diverse interfacce in modo da potersi adattare a differenti scenari di utilizzo. Tuttavia, protocolli obsoleti non permettono di utilizzarle in parallelo e quindi di migliorare le prestazioni offerte all'utente finale. Per colmare questa mancanza sono stati dunque proposti protocolli multi-interfaccia come ad esempio MPTCP che, grazie all'unione delle risorse di più interfacce, promettono di aumentare il throughput e la robustezza. il MPTCP vede le sue prestazioni diminuire nel caso vengano utilizzate interfacce eterogenee ed inoltre ha una complessa gestione delle ritrasmissioni nel caso di perdita di pacchetti. Questa tesi si prefigge inizialmente di studiare un nuovo approccio a questo problema, introducendo dei protocolli multi-interfaccia che utilizzano la tecnica del Network Coding. La particolare struttura del Network Coding permette di risolvere i problemi menzionati prima, di semplificare lo scheduling dei pacchetti tra le diverse interfacce e la gestione delle ritrasmissioni. Successivamente, ci siamo focalizzati sulla gestione dei feedback e delle ritrasmissioni nei protocolli presentati, e in particolare l'impatto che questi hanno sulle prestazioni generali.

Proporremo due protocolli di trasmissione multi-interfaccia, entrambi basati sul Network Coding, e studieremo un schema di feedback compatibile con questi e che permetta di sfruttare le proprietà di questo schema di codifica. Inoltre questi protocolli verrano implementati in un simulatore in linguaggio Python, e una estensiva campagna di simulazioni fornirà i risultati, specialmente riguardo a overhead e feedbacks.

Contents

Listing of figures

Listing of tables

Listing of acronyms

NC Network Coding

RLNC Random Linear Network Coding

ARQ Automatic Retransmission reQuest

FEC Forward Error Correction

RTT Round Trip Time

IP Internet Protocol

TCP Transmission Control Protocol

LTE Long Term Evolution

WiFi Wireless Fidelity

LAN Local Area Network

ETH Ethernet

IETF Internet Engineering Task Force

MP MultiPath

MPTCP MultiPath TCP

RR Round Robin

NC-MPTCP network coding based Multipath TCP

NS Network Simulator

VM Virtual Machine

API Application Programming Interface

TUD Technische Universität Dresden

RFC Request for Comments

1 Introduction

Today's devices that access the Internet are radically different from those of several decades ago, when the building blocks of this technology were developed. At that time, to access the Internet only PCs with a single wired interface were available, and all the transmission protocols such as TCP were designed to work over a single interface. Nowadays, we can access the network through a number of different devices as smartphones, tablets, and notebooks that can adapt to different scenarios in mobility, being equipped with multiple interfaces for different technologies, e.g. WiFi and LTE [6]. Still, they use old protocols that allow to employ only one of them at a time [7]. For these reasons, in the recent years the research and development of multipath protocols have begun that allow to fully exploit the transmission potentialities of these devices. Multipath protocols aim to provide several benefits to the users [8]:

- Effectively increasing of the goodput for the end user, due to the pooling of resource from the different paths
- Vertical handover for mobile devices, that can maintain the transmission active even if one of the links lose the connection, increasing the robustness of the transmission

The most relevant implementation of a multipath protocol is MultiPath TCP (MPTCP) [8], a major extension of traditional Transmission Control Protocol

(TCP) officially released by the Internet Engineering Task Force (IETF) in 2013 [8] (the architectural guidelines were released in 2011 in RFC 6182 [9]), that integrates all the aforementioned characteristics.

The performance of MPTCP as all the other multipath protocols is influenced by a problem called *head-of-line blocking* that arises in presence of heterogeneous links. Works [4] and [5] proposed to solve this problem by introducing Network Coding (NC) in the MPTCP protocol, developing what they call network coding based Multipath TCP (NC-MPTCP).

NC is a promising scheme presented in 2000 [10] that completely redefines the way to transmit the packets in a network. It is detached from the traditional *store and forward* as it encodes packets at network level, adding another level of information encoding. Particularly interesting is the Random Linear Network Coding (RLNC) [11] that aims to encode packets generating linear combinations of them; this scheme has unique features that make it an important block in the telecommunication research [12] [13]:

- Transmission throughput achieves network capacity
- Increased robustness against losses due to its structure that mixes the information of different packets in the same encoded packet
- Ability to generate encoded packets at intermediate node without waiting the decoding; this feature is called *recoding*

These properties led some researchers [4, 5] to implement this scheme in MPTCP because it can solve the negative effects of out-of-order reception and multiple losses that degrade the performance of TCP and MPTCP. With this thesis, we have supported the development of a multipath protocol called `nctun` that, compared to previous works, integrates NC in a more natural and complete way to be used for the transmission of TCP packets over multiple interfaces. The utilization of multiple heterogeneous links to transmit the same session of data introduces further complexity for the scheduling and the retransmission of the lost packets; e.g., a requirement of MPTCP to make the protocol reliable and robust against losses is to allow the retransmission of a packet on a different subflow on which it has been lost. This is not trivial because it require acknowledgements at connec-

tion level but also at subflow level with multiple sequence numbers in the packet header [8].

The main challenge is to *increase the coupling* of the subflows, making the management of a multipath transmission quite equivalent in terms of complexity to the single path counterpart but maintaining all the benefits of exploiting several links. This result can be achieved by exploiting the characteristics of RLNC. RLNC aims to merge the information from multiple packets that belong to a set called *generation* into a single one; this allows the receiver to decode the original information after having received a sufficient number of these encoded packets, as opposed to the traditional *store and forward* that requires the exact set of original packets [13]. For example, a network coded packet received in any instant and from any subflow carries always the same amount of information to the receiver, and can recover the information of one lost packet without requiring the knowledge of a sequence number and the retransmission of a particular packet.

A small modification of the information carried by the ACK packets is needed when NC is employed, as discussed in [14] and [15], that is easy to implement, and as we will see, does not imply a transmission overhead increase with respect to traditional *feedback packets*. In addition, for the reasons mentioned before, a request for retransmission is equivalent to the request for the generation of new random encoded packets. The last feature does not only increases the resilience to packet losses but also allows to *prevent packet erasures* by sending a sufficient amount of redundant packets, if the transmitter has the information about the loss rate of the link.

Most of the research about *feedbacks on network coded transmissions* assumes that feedbacks are transmitted over a *perfect delay-free channels*, and founds the performance evaluation of the protocols onto this assumption. In all likelihood this assumption leads to over optimistic results with respect to the real behavior of the protocol. Real channels are never perfect or without delay and we want to evaluate the impact of the acknowledgement transmission on the final performance. The introduction of this evaluation in a multipath environment underlines a completely different approach with respect to the previous works, which gives a better comprehension of how it is possible to optimize the performance of a network coded multipath protocol and how this performance strictly depends on the quality and reliability of the feedbacks.

In this thesis, two network coded based multipath protocols are investigated. The two protocols differ in the way they employ the available links: the first is suited for *full-duplex links* where the transmission of the packets can be simultaneous with the reception of the acknowledgement packets, while the second is designed for *half-duplex links* where these functions are performed separately. These protocols report two completely different ways to transmit encoded packets and manage feedback and retransmissions, dictated by the channel characteristics. Two aspects of the protocols have been analysed: the performance in terms of throughput and the impact that feedback has on that.

Firstly, the network configurations utilized for the evaluation of the protocols are described. In this set of configurations both single path and multipath networks are present, in order to have a complete comprehension of the behaviour of the protocols in all possible scenarios. Secondly, the developed protocols are explained in detail, and in particular it is explained that we choose to employ only one link for the transmission of the acknowledgement packets also in a multipath scenario, where multiple links could be used. This choice opens the problem about the selection of the *best path for feedback transmission* among the ones available; the search for the characteristics of a path that make it preferable is the main objective of our evaluations.

The aforementioned evaluations are performed implementing the protocols in a simulator specifically developed for this research. All the results are obtained with this simulator that has been written in Python language with the help of Kodo [1], a library with specific tools for the implementation of RLNC protocols.

The rest of this thesis is organized as follows:

- Chapter 2 describes the enabling technology of our protocols, Network Coding and in particular Random Linear Network Coding. In addition, it introduces the main features, issues, and solutions given by other works about *feedback on Network Coding* and *Multipath transmissions*.

- Chapter 3 introduces the Kodo library of *Steinwurf*, describing the main tools and functions provided. Besides, the developed is described in details Python simulator. Third, the feedback management and the developed protocols are introduced.

- Chapter 4 outlines the simulation procedure, presents figures, and comments the results obtained about the two protocols.
- Chapter 5 draws the conclusions and suggests some possible future works.

2

State of art

2.1 Preliminary definitions

In this section we want to introduce some concepts and definitions that will be used later in the thesis.

Network Coding (NC): Usually with the NC term we refer to Random Linear Network Coding (RLNC) [10] [11]. NC is a technique used at network level that acts on packets. This technique breaks-up with the *store and forward paradigm*, e.g. typical of the TCP/IP suite [7], combining several packets together for transmission; we call *encoded packet* the result of the combination of multiple packets. The *encoded packets* in RLNC are generated by linearly combining packets that belong to a set called *generation*. The linear combinations are computed in a *finite field* $\mathbb{F}_q$, where q is the size of the field, and the coefficients are randomly generated: e.g., in $\mathbb{F}_2$ packets and coefficients belong to the binary field (i.e. assume values 0 or 1), and the linear combination of two packets correspond to the XOR (logical exclusive OR) of them. The coefficients of the linear combination are stored in the *header* of the *encoded packet*. The receiver collects the *encoded packets* in a buffer and fills a matrix (called *decoding matrix*) with the received coefficients of the linear combinations; when the receiver has received enough packets it performs the decoding operation and recovers all the

packets that belong from the current *generation*. NC improves the transmission throughput, theoretically reaching the capacity of a network, and the resilience to losses [12] [13].

Erasures: The system developed in this thesis exchanges information in the form of packets at network level. In the ISO/OSI model [16] the protocols at network layer rely on the services offered by lower layers, that aim to mask issues due to noise, interference and fading at physical layer [17]. What is experienced at network layer is the so called *packet erasure channel*, where the packets are either successfully received or lost. There are other events in a packet network, e.g. congestion or full buffer, that imply an erasures. In this thesis we refer to erasure and loss as the event of failed reception of a packet.

Perfect channel: In this thesis we call *perfect channel* a link with zero-delay, probability of erasures equal to zero and infinite bandwidth. Every packet that is transmitted through a *perfect channel* is instantaneously delivered at the receiver with probability 1.

2.2 Network Coding

The term network coding has been coined in the paper [10] referring to "coding at a node in a network as *network coding*". No particular coding scheme is specified, but the term applies map from input packets to outputs. This definition of network coding is the most general possible because it does not depends on a particular coding scheme, and reveals the flexibility of a novel concept of coding. Network coding in this first paper idealizes, in a way, the usual information theory where each node in a network has a probabilistic effect on other nodes, because they define *network coding* in a network where nodes are interconnected by error-free point to point links. This assumption does not seem to be very realistic, if we think of present-day connections, wireline connections may approach this condition, but wireless channels are more challenging. The network in the model of [10] is in fact an abstraction of physical layer connections, approaching the concept of network layer connection. If physical layer is not able to provide a reliable connection to the higher layers, the reliability can be achieved with data link control, network, and transport layers. Then, the first citation that we have mentioned can be extended leading to the very first description of *network coding*:

coding at a node in a network with error free links [13]. The absence of lossy links suggests that network coding is not a channel coding procedure that recovers data corrupted by a noisy channel. Channel coding is an arbitrary mapping from received input symbols to outgoing channel symbols, from a source node that performs the encoding to one or more destinations, where symbols encoded travel over the physical layer. Channel coding is applied to physical connections, sources and destinations of this type of encoding have to be directly linked via the physical medium between them [13] [18]. Network coding proceeds in the same manner of channel coding, but exploiting the features of higher layer protocols on top of the physical links. Nodes that communicates with a network coded transmission have not to be physically and directly linked, but simply have to be logically connected in the higher layer network. Given the specifications of the network, its behaviour and the probability function of the erasures of packets its possible to define capacity as the maximum reliable rate that can be achieved. The capacity that we have defined is measured in packets per unit time and can be achieved with network coding as we will see later. The main difference between these two procedures is that channel coding maps symbols of the physical channel, network coding is an arbitrary mapping function of the content of each node's incoming packets. Packets at upper layers have to provide robustness against erasures, while channel symbols are intended to be robust against noise. Secondly, packet level encoding allows to append side information in the header of the packet. Thirdly, due to processing, queueing and dropping procedures at lower layers, packets transmission are not synchronized as can be intended the symbol transmissions.

The usage of a higher level representation of the information allows to have a better view of the network, giving the possibility to manage not only the single transmission but also the interaction of multiple flows exchanged through the network, and thus also the global behavior of the system and its performance.

The usual way to build a fully reliable communication at packet level is to use a feedback protocol as the *Automatic Retransmission reQuest* (ARQ) [17], that can be implemented in a link-by-link setup or in a end-to-end one, or both. The optimal case where the capacity can be achieved using network coding is when we have an uncongested network with perfect zero-delay feedbacks. Congestion is in fact a specific problem of packet level networks that causes packet dropping

and increased delay in a transmission; furthermore, unreliable and slow feedbacks can represent a very important problem for an ARQ scheme [19] [20]. Another scenario where end-to-end ARQ could be not well-suited in usual packet networks is the multicast transmission, where each node that experiences a packet loss notifies the source, possibly overloading the network; in addition, the retransmitted packets are probably useful only for a subset of the nodes. Network coding allows to use a feedback and retransmission control drastically different from the usual ones.

Network coding thus is not a single precise coding scheme but a general new approach that introduces a set of coding opportunity in a different domain, at packet level. Packets are different from physical layer symbols for the reasons that we have previously explained. Symbols are simply transmitted from a source to one or more destinations in an end-to-end way through a physical channel, packets are exchanged in a more complex scenario with multiple nodes to go through to reach the destination, in a network where multiple users want to transmit and then, sharing the same links with other users.

A subset of network coding techniques that are thought in order to increase the performances of transmission from a single user are called *intra session network coding*, that aims to encode packets that belong to the same transmission session; the most important coding scheme that is part of this family is the *random linear network coding* [12] that is usually confused with the more general term of *network coding*. Random linear network coding is the most used and famous network coding technology because it has several interesting properties: it is able to approach capacity of the network, it is decentralized then it does not need coordination between nodes, and in particular it is a rateless code i.e. from the same set of source packets an infinite number of coded packets can be generated [13] [12]. This is particularly useful in lossy network where the loss rate could change over time.

In the *multi-unicast* case, where in the same network there are $N > 1$ sessions between a set $S = \{s_1, s_2, \ldots, s_N\}$ of sources and a set $D = \{d_1, d_2, \ldots, d_N\}$ of destinations [21], there is another type of network coding techniques that can be used called *inter flow network coding* [22], that aims to encode at a node in the network packets that arrive from different sessions. *Inter-session network coding* is more complicated than *intra-session network coding* because it needs a complex

management of the coding procedures at intermediate nodes in order to guarantee that each sink can decode its desired source process.
In the following section we will analyse the coding scheme that we have employed in our research: *Random Linear Network Coding*.

2.2.1 Random Linear Network Coding

Random linear network coding is a technique that aims to encode the packets that belong to the same transmission session. The main idea of this scheme is that a node, instead of simply forwarding its packets individually, sends coded packets that contain a linear combination of the information inside of the original packets. Each packet that carries a linear combination of other packets also has to piggyback side information about the particular combination in the payload, and this operation is made possible and simple because we are at the packet level and then we can store side information in the header of the packet. Then the main difference between a usual coding scheme that maps an input symbol to an output one is that here we jointly code multiple packets in a single one, carrying more information than a normal forwarded packet.
In order to better understand the important advantages that such a coding scheme brings with it we mathematically formalize the procedure. At the source node we have K uncoded packets $w_1, w_2, \ldots, w_K$ which are vectors of length λ with values over the finite field $\mathbb{F}_q$ (that can be also called Galois field of size q). Practically, the information that the source has to transmit is split into K segments of the same length λ where the set of this packets $w_1, w_2, \ldots, w_K$ is called *generation*. If the original packets have a length of b bits, then each of the K segments of the generation will have a length of $\lambda = \lfloor b/\log_2 q \rfloor$ in the Galois field of length q.

The coding procedure at the source node is simply the creation of a linear combination among the K packets of the generation, creating an outgoing coded packet with the same length of the uncoded packets (except for the header that has to contain side informations) and that has inside coded information of potentially all the K packets. The cases in which is created a linear combination of a subset of packets is included in the general case taking the coefficients corresponding to the packets not in the subset equal to zero. Since the whole procedure is linear

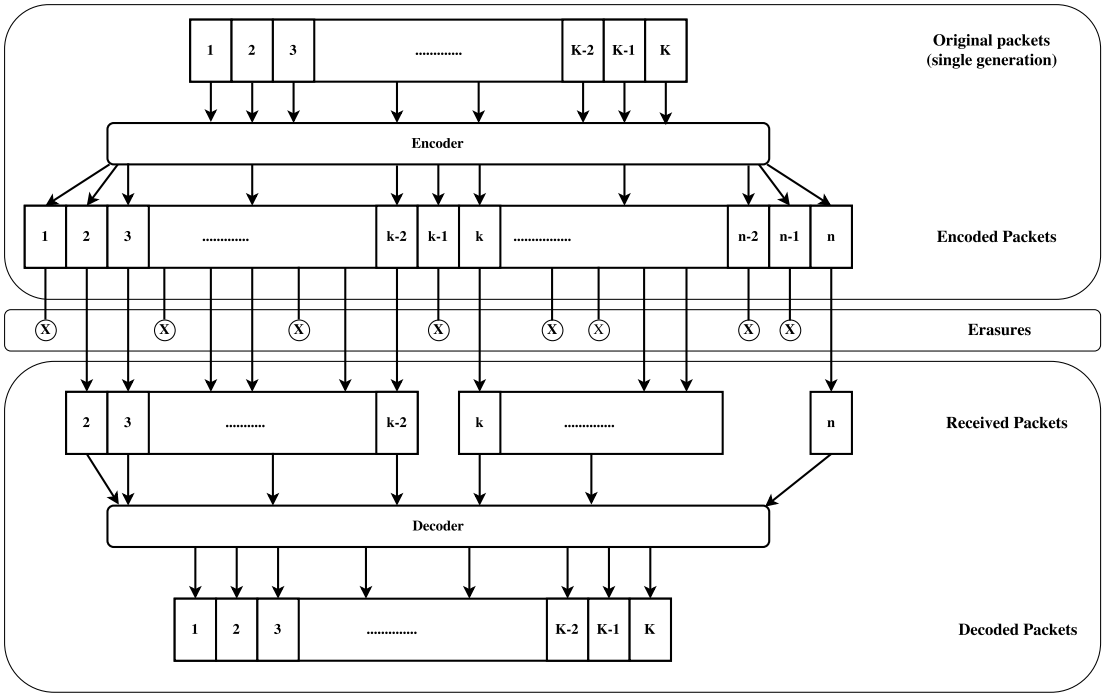


Figure 2.1: Overview of the Random linear network coding procedure[1].

we can describe any coded packet x as a linear combination of $w_1, w_2, \ldots, w_K$, thus

$$x = \sum_{k=1}^{K} \gamma_k w_k \tag{2.1}$$

that can be written in matrix form $\mathbf{X} = \mathbf{\Gamma} \cdot \mathbf{W}$. When a coded packet is created at the source in the header is stored the vector of coefficients $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \ldots, \gamma_K]$ that is fundamental for the decoding operations, the overhead of $K \log_2$ bits is negligible if packets are sufficiently large. In Fig. 2.1 we can see the coding procedure where the K packets of the generation are passed through the encoder that generate n encoded packets that are linear combinations of the original ones. A destination node collects the received packets and, if it has received K packets with linearly independent coefficients vectors then the node is able to decode the K original source packets, inverting the matrix $\mathbf{\Gamma}$ i.e. $\mathbf{W} = \mathbf{\Gamma}^{-1}\mathbf{X}$. The decoding operations are made by Gaussian elimination [23] [24]. The *random* part of the procedure is in the encoding, in fact the coefficients vectors $\boldsymbol{\gamma}$ are chosen randomly in $\mathbb{F}_q^K$ generating random linear combinations between the packets of the same

generation. The operation of choosing randomly a vector of coefficients can be made infinite times, meaning that it is possible to generate a infinite number of coded packets from the same set of source packets, for this reason this set is called *generation*. *Random linear network coding* is then said to be a *rateless code*, since there is no predefined *rate* of the code but we are able to generate how many encoded packets we need, and this is one of the most important feature of this coding scheme. In Fig. 2.1 the number of generated encoded packets $n \geq K$ is reported in lower case to underline that this is not a predefined number but can be chosen a posteriori in function of the erasures of the medium. At the decoder we assume to have received a number $r \geq K$ of linearly independent coded packets, and then passing those through the decoder that applies the Gaussian elimination we recover the original data.
Since the decoding of the data depends on the reception of a number $n > K$ of linearly independent combinations then the probability that a source generates a linear dependant combination becomes an important factor. We denote with $K' = K - \widetilde{K}$ the number of packets needed by a sink in order to decode, where $\tilde{K}$ is the number of linear independent packets already received by the sink. This interesting point has been analyzed in [25]. For a given q and K the probability that a generated combination is linear dependent to another one can be written as

$$P_{dep} = 1 - \prod_{i=1}^{K}(1 - \frac{1}{q^i}) \tag{2.2}$$

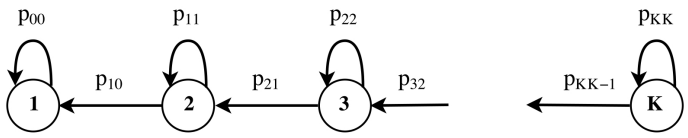


Figure 2.2: Markov Chain for a single node that receives K packets.

A sink receiving K packets can be represented as a Markov chain as reported in Fig. 2.2, where the probability that a packet is useful at a sink is the probability it is received multiplied with the probability that it is independent

$$P_{i\to(i-1)} = (1-p)\left(1 - \frac{1}{q^i}\right) \tag{2.3}$$

The probability that a packet is not useful can be calculated as the complementary of the previous one, that is the probability that the packet is not received plus the probability that is received but linear dependent from the ones already received

$$\begin{aligned} P_{i\to i} &= 1-(1-p)\left(1-\frac{1}{q^i}\right) = 1-1+\frac{1}{q^i}+p+p\frac{1}{q_i} = \\ &= p+(1-p)\frac{1}{q^i} \end{aligned} \tag{2.4}$$

Obviously the more packets the sink has collected, the higher the probability that a new received packet is not useful, but from 2.4 we can also see that the dimension q of the finite field that is employed plays an important role in the value of this probability. In fact a field with a bigger size guarantee a lower probability of generating linearly independent packets at the price of an increased computational complexity of each operation in the coding procedure. This value impact on the efficiency of thee coding procedure because a higher probability $P_{i\to(i-1)}$ require an higher mean number of transmitted packets. If we denote $\mathbf{C}$ as the transition matrix of the Markov chain in Fig. 2.2

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ (1-p)\left(1-\frac{1}{q^1}\right) & p+(1-p)\frac{1}{q^1} & \dots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & (1-p)\left(1-\frac{1}{q^K}\right) & p+(1-p)\frac{1}{q^K} \end{bmatrix} \tag{2.5}$$

Then the expected number of transmission for one packet of the total K packets can be found by

$$E[t_x] = \frac{1}{K}\sum_{i=0}^{\infty} 1-\mathbf{C}^i_{(K,0)} \tag{2.6}$$

Then we have seen that the choice of K and q can have an impact on the efficiency of the code and then we have to find a good tradeoff between efficiency and complexity.

Since we are coding at the packet level, each packet that the source transmits can go through multiple nodes before reaching the intended destination. Intermediate nodes can act in two different ways: only *forward* the packets that it receive to the next node in the routing table or *recode* the packets. In the *recode* setup,

when an intermediate node receives a packet, it stores the packet in the internal memory. Every time that an erasure happens on an outgoing link of the node, the latter can generate a packet that is a linear combination of the previously received packets. Supposing that the node has L packets stored in the memory, for example $y_1, y_2, \ldots, y_L$, then the node can produce the outgoing packet:

$$y_0 = \sum_{l=1}^{L} \alpha_l y_l \tag{2.7}$$

with α_l chosen according to a uniform distribution over the elements of $\mathbb{F}_q$. Due to the structure of the packets this new packet y_0 is itself a linear combination of the original packets w_k of the generation

$$y_0 = \sum_{k=1}^{K} \left(\sum_{l=1}^{L} \beta_n \alpha_{nk} \right) w_k = \sum_{k=1}^{K} \gamma_k w_k \tag{2.8}$$

, and the coefficients $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_K\}$ are inserted in the header of y_0.

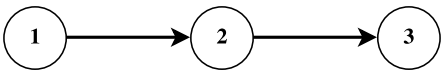


Figure 2.3: Two-link tandem network.

The opportunity of coding at intermediate nodes is a very important feature in the case of a network with lossy links, as seen in the following example [13]. In Fig. 2.3 we have the node 1 that wants to transmit some packets to node 3 passing through 2. Packets are lost, i.e. the entire packet experiences an erasure, on the link between nodes 1 and 2 with probability ε_{12} and on the other link with probability ε_{23}. With an end-to-end erasure code from node 1 we can achieve the maximum rate of $(1-\varepsilon_{12})(1-\varepsilon_{23})$. The true capacity of the system is not achieved; if we use two different stage of encoding, i.e. enabling recoding at node 2, we are able to communicate between nodes 1 ad 2 at rate $(1-\varepsilon_{12})$ and between nodes 2 and 3 with rate of $(1-\varepsilon_{23})$ packets per unit time. Thus, this strategy allows us to communicate from node 1 to 3 at a rate of $\min\{1-\varepsilon_{12}, 1-\varepsilon_{23}\}$ that is in general greater than $(1-\varepsilon_{12})(1-\varepsilon_{23})$. It is possible to employ this strategy with every erasure code, simply decoding and recoding the received packet at each node, but it is not a feasible option because the decoding procedure at each intermediate

node increases the delay too much. The unique feature of *random linear network coding* is that intermediate nodes are able to generate encoded packets also with an incomplete set of information and *without decoding* the incoming packets [10] [26] [27].

Now that we have briefly analysed the increased robustness to erasures with network coding we can go further to see which are the throughput gains that we can achieve. To employ network coding can provide some throughput gain in different scenarios, like the ones that we have described in the introduction: *unicast, multicast and broadcast*. A *multicast* scenario where the network coding can achieve the capacity of the network is the *butterfly network*, reported in Fig. 2.4 and presented for the first time in [10].

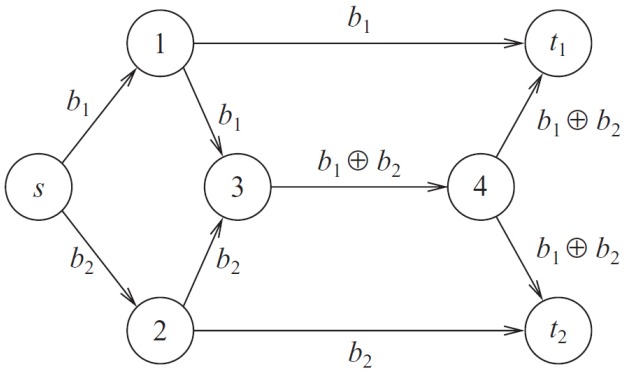


Figure 2.4: *Multicast scenario*: butterfly network.

The Galois field used in the example network has size equal to 2, meaning that each coding operation between two or more packets results in a XOR between them. This example is not directly related to random linear network coding because in the coding operations the coefficients are not chosen randomly, but it shares most of the property and then the results are still valid for *random linear network coding*. Source node s wish to send packets b_1 and b_2 to both sinks t_1 and t_2. The multicast transmission can fully exploit the *capacity* of the network only if one of the intermediate nodes does not only forward the packets but decides to network code them. In the example Node 3 receives both packets b_1 and b_2 then following the same algorithm explained for the recoding operation in random linear network coding, it can produce a coded packet that contains the bitwise XOR of both packets. All the other nodes only forward the received packets. Each

of the two sinks finally receive two linearly independent packets, one uncoded from the side link and one encoded from the Node 4, then they can decode both the original packets. The nine packets used in the butterfly network carry the information of two packets, but without coding its not possible to transmit as much data and other transmissions are needed. The *butterfly network* example is contrived and rarely could represent a real network but is useful to show how the utilization of a simple network coding scheme can increase the throughput in a multicast wireline network. The throughput benefits are not restricted only for wireline networks, but can exploit also the broadcast nature of wireless networks.

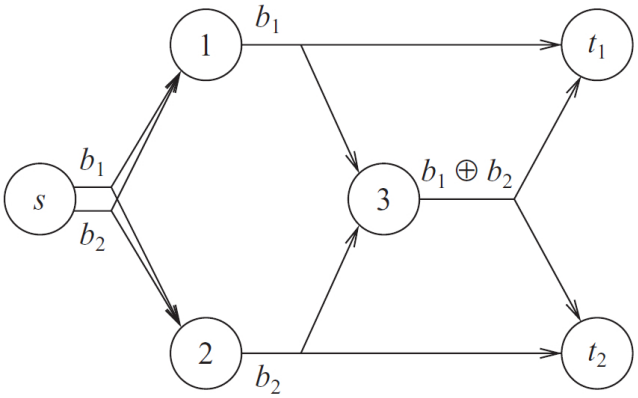


Figure 2.5: *Wireless multicast scenario*: wireless butterfly network

Fig. 2.5 represents a network similar to the one used in the previous example with the difference that in this case the arcs of the graph represent wireless transmissions; in fact each transmission is not heard only by one node but also by all neightbors of the source. Also in this case only node 3 network codes the received packets. Exploiting the broadcast nature of the wireless medium, Node 3 is able to transmit the coded packet to both sinks simultaneously, and this packet is useful for both. In this case node 3 reduces by 1 the number if transmissions with respect to the case without network coding where the same node has to transmit b_1 and b_2 separately. On the other side, also the receivers t_1 and t_2 have the advantage of not receiving useless packets intended for the other sink.

Wireless links, in addition to the broadcast nature, have the important characteristic to be subject to interference, fading and collisions, thus at the packet level wireless links are seen as lossy links [17]. Sometimes packets are lost during

the transmission and the intended sink could require a retransmission of these packets. We put ourselves in a *broadcast* scenario where a single source wants to transmit the same information to N receivers. Each receiver can experience a different quality of the channel at different times, then a single sink could not receive packets that other sinks have correctly received. In a normal packet network that employs an ARQ scheme, each sink sends a request of retransmission of the packets that has not received, possibly overwhelming the source node of requests and of a different retransmission schedule for each user. In addition, the sinks will probably receive useless packets that are not intended for them. With NC all the received packets are useful for several users due to the linear combination structure.

2.3 Feedback

In the previous section, we have talked about the benefits that network coding can bring in a packet network. In a *multicast scenario* we know from [10] that the source can transmit data at a rate arbitrarily close to the capacity of the network, by allowing intermediate nodes to randomly combine their incoming packets. These results are formalized in the coding theorems also in [10], that are like bounds that we can achieve in ideal conditions. These results assumes the complete knowledge by the source of the min-cut capacity of the network and that it will code information at that rate. Furthermore, they assume that the nodes of the network are *benevolent users* that aims to maximize the global performance of the system, and also that there are no complexity constraints. This last assumption means that all the intermediate nodes can encode all their incoming information flows with an arbitrarily complex Forward Error Correction (FEC) scheme. This approach is investigated in [28], [29], [30] and others, employing low complexity codes at network or application layer.
The utilization of feedback schemes instead of FEC codes is an alternative way to proceed that can bring benefits but also complicates the management of the network. Usually feedbacks are assumed as ideal, meaning that the source knows instantly if its transmission has been successful or not. In addition, idealized feedbacks guarantee to achieve capacity of the network. In real networks feedbacks are not free of costs because they have the same characteristics of other packets in

the network, then suffering of delay, congestion, and erasures, in practice they are a new source of traffic that has to be reliably delivered. Despite these problems, in a Network Coding enviroment feedbacks are fundamental to achieve *reliability against losses* and *parameter adaptation* [14]. The *parameter adaptation* is an important procedure that has to be implemented in order to allow the code to achieve the best possible performance. For example we can consider the parameter of the *size of the generation* that in the previous section we have denoted as K. The size of the generation can affect the rate that a sink can experience and the error correcting capability of the code [25]. The first because a bigger generation size leads to bigger coefficients vectors in a Galois field of higher dimension, thus, with a lower probability to have linear dependent vector. The second because, in a multicast scenario, with a higher K is more probable that a single packet can be useful to several sinks in order to recover losses. These advantages comes at the price of a higher computational complexity and decoding delay. The complexity is increased due to the greater size of the coefficients matrix to invert during the decoding procedure, while the delay is affected by the higher number of coded packets that a sink has to wait until decoding. The *trade-off* is between a higher erasure robustness and throughput versus a delayed decoding. Feedbacks from the sinks can then be useful for the source in order to optimize the *generation size* in function of the current channel quality.
We can think an example of a delay constrained transmission where packets received after the time T_{thres} are useless and then dropped. The network setup is reported in Fig. 2.6 with h sources, $N = \binom{n}{k}$ receivers and k coding links $\{A_i, B_i\}$. Each of the N receivers R_i observes a distinct subset of h B nodes. Time is slotted and in every slot each source can transmit one packet through every link, that have unitary capacities. The network has min-cut capacity h at the source and between nodes B_i and the receivers R_i. The single source S underlines that the packets are all from the same session than we are talking about *intra session network coding*. We assume that the edges between nodes A and B are associated with a delay that changes in a i.i.d. fashion according to a probability distribution $\mathbf{P}$; in addition, the delay changes at a lower time rate, e.g. they changes each M time slots.

In order to transmit at rate h i.e. at the mincut capacity to all the receivers, the source has to send at most h uncoded packets and $k - h$ packets coded

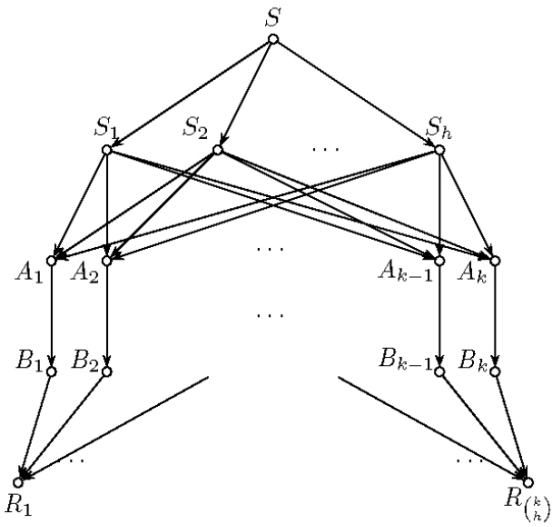


Figure 2.6: Network with h source and $N = \binom{n}{k}$ receivers.

through Random Linear Network Coding. In order to decode in a useful time a receiver has to receive the packets from the h links that observes with a delay lower than T_{thres}, that happens with a probability of $P(d < T_{thres})^h$. With only one of the edges that exceeds this threshold all the received packets at a sink become useless. If instead the sources send only uncoded packets then we have not problem of decoding delay meaning that a link with a delay that exceeds the threshold does not invalidate all the previously received packets because do not need to be decoded. On the other hand, sending only uncoded information does not guarantee to achieve min-cut rate at all the receivers and there could be sinks that receives packets at a rate equal to 1; this happens when a receiver observes the same uncoded packets from all its h links.

If the source receives feedbacks from the receiver nodes it can decide of not use the links with a high delay (this is possible due to the assumption of the slow changing of the delay of the links) and modify the coding scheme in order to maximize the transmission rate on the remaining links.

The last example shows us how the feedbacks are important in order to adapt the coding scheme to the network where it is applied, optimizing the throughput and the delay. The parameter adaptation is not the only advantage of the utilization of the feedbacks with network coding. Now we want to report some results from [14] that aim to prove how the usage of the feedbacks can help to save resources, in

systems that combines in different ways ARQ and FEC. These schemes are applied on the network in Fig. 2.7 where the source A aims to transmit information to node C through the intermediate node B. The link between nodes A and B has a loss rate of ε_{AB} and the other one ε_{BC}. Assuming $\varepsilon_{AB} < \varepsilon_{BC}$ the mincut capacity is equal to $(1 - \varepsilon_{AB})$.

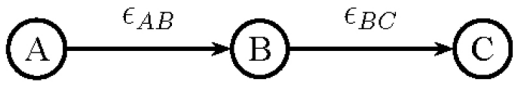


Figure 2.7: A path from source A to Node C through intermediate Node B.

In Table 2.1 are reported the characteristics and the results of the five employed coding schemes. The reported values are:

1. *Rate*: the maximum achievable rate at which information can be transmitted from A to C.

2. *Delay*: represent the excess time $(d - k/C_{mc})$ with respect to the theoretical minimum that node B takes to transmit k packets; C_{mc} is the mincut capacity and d is the time instant when B finishes transmitting.

3. *Feedback*: the number of feedback packets required by the coding scheme

4. *Memory*: the quantity of memory needed at node B

5. *Blocksize*: The dimension of an single coding block. Bigger coding block correspond to larger delay.

When employed, feedbacks are assumed lossless and instantaneous.

Schemes		Rate	Delay	Feedback	Memory	Blocksize
I	FEC	min-cut: $(1 - \varepsilon_{AB})$	$O(\sqrt{k \log k})$	0	$O(k)$	k
II	End-to-end FEC	$(1 - \varepsilon_{AB})(1 - \varepsilon_{BC})$	$O(k)$	0	0	k
III	Fec-m	$(1 - \varepsilon_{AB})(1 - \pi_m)$	$O(k)$	0	m	k
IV	ARQ $(R_f = 1)$	min-cut: $(1 - \varepsilon_{AB})$	$O(\sqrt{k})$	k	$O(\sqrt{k})$	1
V	FEC + ARQ (R_f)	min-cut: $(1 - \varepsilon_{AB})$	$O(\sqrt{k})$	kR_f	$O(\sqrt{k})$	k

Table 2.1

- *Scheme I*: In this scheme Nodes A and B use a capacity achieving FEC code. Node A uses a code C_1 to encode and transmit n_1 packets over the link AB. Node B receives on average $n_1(1-\varepsilon_{AB})$ coded packets after n_1 time slots, decodes (or recode without decoding if possible) and transmits n_2 coded packets with the code C_2. If Node B finishes to transmit at time d, Node C will receive on average $n_2(1-\varepsilon_{BC})$.

- *Scheme II*: This scheme is a simplified version of the previous one, because the code is applied only at Node A, and Node B only forward the packets that it receives. This scheme does not require processing and memory capabilities at Node B but does not achieve capacity.

- *Scheme III*: It is similar to the first scheme, with a constrained memory at the Node B that can contain at most m packets. This scheme collects the received packets in the memory and outputs a stream of packets that are linear combination of those. It achieve only $(1-\pi_m)$ of the mincut capacity, where $\pi_m < \varepsilon_{BC}$ is the steady state probability of an appropriately defined Markov chain as described in [30].

- *Scheme IV*: Ensures the correct reception with one feedback for each packet (i.e. at rate $R_f = 1$). This scheme achieves the mincut capacity but also overwhelms the network of feedback packets. It is important to note that this scheme can reach this performance only if the feedbacks are assumed perfect, i.e. with zero-delay and no erasures.

- *Scheme V*: Here a block coding scheme is employed(e.g. RLNC) with in addition a feedback protocol to notify the correct reception of a block. Every block has length equal to k then a feedback is transmitted at an average rate of $1/(k(1+\epsilon))$ that is the rate at which the receiver collects the k packets useful to decode the block. We can generalize the rate of feedback

transmission introducing R_f:

$$R_f = \frac{\#\text{ of feedback msg}}{\#\text{ of received pcks}} \tag{2.9}$$

where a rate of $R_f = 1/f$ means that a feedback packets is transmitted every f time slots. The scheme uses f times less feedbacks than the previous one, requiring an average memory at node B of $1/R_f(1-\epsilon_{BC})$ larger than that with ARQ. The main benefits is that we are able to reach the mincut capacity requiring feedbacks only on the link BC instead of both AB and BC as in the scheme IV.

This example shows how feedbacks can improve performance of the transmission when is combined with a block coding scheme as *random linear network coding*, referred to a better memory management delay. If we focus our attention to the case where TCP is used to ensure reliable communication the application of ARQ is not straightforward. This case is discussed in [14] and [2] that report similar solutions to the problem.
In RLNC every received packets is formed by the linear combination of packets that belong to the same generation. A node is not able to perform a full decoding of the original packets until it has correctly received K linear independent packets. If we follow a batch based approach, the packets are acknowledged only when the full decoding is reached, leading to a decoding delay that increase as a function of the generation size K. This approach could interfere with the TCP retransmission mechanism that would either timeout or observe a high round trip time leading to a low throughput. Paper [2] aims to build an ACK-based sliding-window network coding approach compatible with TCP. TCP uses acknowledgements for notify newly received packets as they arrive in *correct sequence order*, in order to guarantee reliable communication and also as a feedback for the congestion control loop. The difference with the normal TCP is that the receiver does not observe a packet but a linear combination of packets that has not something that can distinguish it from the other coded packets. What is missing is the notion of *ordered sequence* of packets as used in TCP because at the source we are mixing the originally ordered packets. In addition, a received coded packets could be linearly dependent on those already received, not carrying new information to the

receiver. If we think the decoding problem at the receiver as a system of equations in K unknowns, every time a new useful packet is received a new equation can be inserted in the system subtracting a degree of freedom from the system. Thus, a new unit of information corresponds to a *degree of freedom*, then acknowledging each degree of freedom can be a strategy compatible with TCP. We now want to formalize this intuition about the degrees of freedom, as reported in [2]. We consider a set of K packets $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_K$ that are vectors over the finite field $\mathbb{F}_q$. The k^{th} packet in the source generation is said to have a *index* k. A node that performs network coding can for example produce packets $\mathbf{q}_1 = \alpha\mathbf{p}_1 + \beta\mathbf{p}_2$ and $\mathbf{q}_2 = \gamma\mathbf{p}_1 + \delta\mathbf{p}_2$, where $\alpha, \beta, \gamma, \delta \in \mathbb{F}_q$. Assuming the packet length equal to l and $K = 2$ the two packets can be written in matrix form

$$\begin{pmatrix} q_{11} & q_{12} & \ldots & q_{1l} \\ q_{21} & q_{22} & \ldots & q_{2l} \end{pmatrix} = C \cdot \begin{pmatrix} p_{11} & p_{12} & \ldots & p_{1l} \\ p_{21} & p_{22} & \ldots & p_{2l} \end{pmatrix} \tag{2.10}$$

where $C = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$ is the coefficients matrix. At the decoder side it has only to invert matrix C in order to recover original data. In this setting [2] introduced the following definitions.

Definition 1 (Seeing a packet)**.** *A node is said to have seen a packet* $\mathbf{p}_k$ *if it has enough information to compute a linear combination of the form* $\mathbf{p}_k + \mathbf{q}$*, where* $\mathbf{q} = \sum_{l>k} \alpha_l \mathbf{p}_l$*. Thus,* $\mathbf{q}$ *is a linear combination involving packets with indices larger than* k*.*

Then a node *has seen* a packet if it has received at least one linear combination where the coefficient relative to the k^{th} element is different from 0. If the node has also *decoded* $\mathbf{p}_k$ then it can also compute a linear combination $\mathbf{p}_k + \mathbf{q}$ with $\mathbf{q} = \mathbf{0}$.

Definition 2 (Knowledge of a node)**.** *The* knowledge of a node *is the set of linear combinations of original packets that it can compute, based on the information it has received so far. The coefficient vectors of these linear combinations form a vector space called the* knowledge space *of the node.*

Proposition 1. *If a node has seen packet* $\mathbf{p}_k$*, then it knows exactly one linear*

combination of the form $\mathbf{p}_k + \mathbf{q}$ *such that* $\mathbf{q}$ *is itself a linear combination involving only* ***unseen*** *packets.*

From this proposition derives the following definition

Definition 3 (Witness)**.** *We call the unique linear combination guaranteed by Proposition 1 the witness for seeing* $\mathbf{p}_k$*.*

In Fig. 2.8 is represented the basis matrix of a node in a row-reduced echelon form (RREF) that is the form obtained by applying Gaussian elimination on the coefficient matrix. The *seen packets* are the ones that corresponds to the pivot elements of the basis matrix, the *unseen* elements are the ones that have not still a row with the corresponding pivot element.

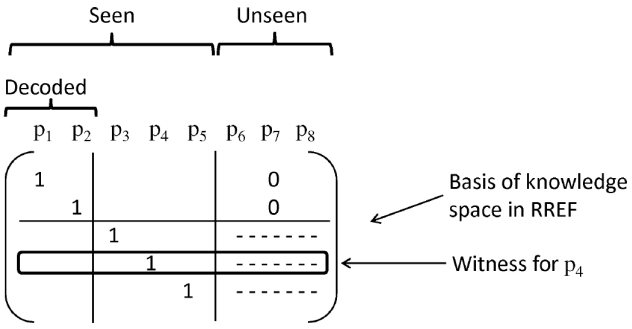


Figure 2.8: Seen packets and witness in terms of the basis matrix [2].

The main observation made in [2] is that the number of seen packets is always equal to the dimension of the knowledge space, that is equivalent to say that is equal to the number of degrees of freedom received at the moment. A new packet that increases this number is said to be *innovative*. If the field size is very large with high probability every newly received packet is *innovative*.
This observation leads us to a feedback mechanism for network coding transmissions on TCP, that consists in sending an acknowledgement packet for each *innovative packet* received. In particular, the strategy proposed is this [2]:

1. The sender takes packets from the TCP source and put them in an encoding buffer. The encoding buffer represents the set of packets from which are generated the linear combination, when a packet has been acknowledged it is deleted from the coding buffer.

2. For each packet R linear combinations are sent to the IP layer on average. R represent the redundancy parameter. The redundancy is an important parameter because if we send a low number of packets, then due to losses, the receiver will not be able to decode, requiring some retransmissions. This is a situation where the losses are not masked to TCP, resulting in a higher Round Trip Time (RTT) (i.e. the amount of time that the a packet takes to be sent plus the amount of time that it take to be acknowledged at the transmitter) and lower throughput. Also a higher R is not a good choice because too many packets will congest the network.

3. When the receiver receives a linear combination, it extracts the coefficients from the header and put them in the decoding matrix. Then it performs Gaussian elimination on the matrix to find out the newly seen packet. If there is one, an ACK is created and transmitted to the source. The receiver maintains a buffer with the packets that are not yet been decoded.

The mechanism proposed is then to send a feedback for each *innovative* coded packet that is received, that contains the index in the generation of the newly *seen* packet. The idea of the sending algorithm that exploit the information carried by the feedback packets is a sort of *sliding window* coding. In fact, the source does not generate linear combinations of all the packets of the generation but only of those that are inside a *coding window*. The *coding window* at the source has a defined length and shifts of one packets when the source receives a feedback from the receiver that notifies the last seen packet. Fig. 2.9 reports an example of the behaviour of this technique in a multicast scenario with a source and two receivers A and B. The source has to transmit a generation composed by packets $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$ and the window length is equal to 2, leading to linear combination that are at most composed by two packets.

Time	Sender's queue	Transmitted packet	Channel state	A		B	
				Decoded	Seen but not decoded	Decoded	Seen but not decoded
1	p_1	p_1	$\rightarrow$ A, $\nrightarrow$ B	p_1	-	-	-
2	p_1, p_2	$p_1 \oplus p_2$	$\rightarrow$ A, $\rightarrow$ B	p_1, p_2	-	-	p_1
3	p_2, p_3	$p_2 \oplus p_3$	$\nrightarrow$ A, $\rightarrow$ B	p_1, p_2	-	-	p_1, p_2
4	p_3	p_3	$\nrightarrow$ A, $\rightarrow$ B	p_1, p_2	-	p_1, p_2, p_3	-
5	p_3, p_4	$p_3 \oplus p_4$	$\rightarrow$ A, $\nrightarrow$ B	p_1, p_2	p_3	p_1, p_2, p_3	-
6	p_4	p_4	$\rightarrow$ A, $\rightarrow$ B	p_1, p_2, p_3, p_4	-	p_1, p_2, p_3, p_4	-

Figure 2.9: Example of sliding window.

This coding scheme allows to minimize the decoding delay because sending packets on a small window allows to decode when all the packets depending from that window have been *seen* from the decoder, and not when at least K innovative packets are received. In addition, the buffer required at the decoder is smaller because it is useful to store only the packets of the current windows that are *seen but not decoded*, because when they are decoded they are passed to the TCP layer. In this way, the TCP layer does not suffer of the high RTT and the reception of the packets is perceived as in order. It is important to note as these improvements on the performance of the random linear network coding scheme are possible only with a feedback mechanism that provides the information about the state of the receivers at the source. As in the previous example, the mechanism can be simply applied to multicast scenarios where network coding can manifest all its benefits, and not only to the classical unicast TCP connections.

What we have seen is a protocol that allows to apply random linear network coding to the standard TCP inserting a network coding layer that implements the retransmission algorithm between IP and TCP layers. This permits to exploit the performance gains of RLNC without modifying the TCP protocol, fundamental if we think about an implementation in a real network. Not always a complex mechanism is required, when we do not have memory and decoding delay constraints. In these cases it is possible to see how Network Coding can simplify also the feedback management. Usual ARQ schemes as *Go Back-N* are *selective repeat* and not still applicable with Network Coding due to the lack of the *sequence ordering* that we have previously explained. In order to request a retransmission to the source, the receiver has to notify how many linear combinations it has received, and not the exact packet that has been lost on the erasure channel.

2.4 Multipath

In recent years with the emergence of wireless connections to access the Internet the number of devices with multiple interfaces is increased. Few years ago the usual way to connect to the the Internet was a PC with a wired connection, now we have devices as the smartphones that are provided with LTE and WiFi [6] and notebook that have both wired connection (Ethernet) and wireless connections (WiFi and sometimes also LTE). For the users of these devices seems to

be logical to improve the network experience and performance when connected through multiple interface instead of a single one. The expectations of the users in this case do not have a straightforward solution for the real implementation. The main problem in the realization is that the most used transport protocol (TCP) is not suited to scale on multihomed devices. TCP is a protocol that aims to build a tunnel through the network between two network interfaces, then it is not possible to adapt the protocol without deep modifications. In order to tackle this problem the IETF released a dedicated version of TCP under the name of MPTCP, which was inserted for the first time in the Linux kernel on July 2013, and its development is still ongoing. MPTCP is an extension of TCP presented for the first time in [31] and [32] and then officially inserted by the IETF in Request for Comments (RFC) 6824 [8]. Pooling the resources of multiple channels allows to increase the throughput with respect to TCP. In a wireless scenario, as the aforementioned example of the smartphone, MPTCP can dynamically manage the channel interfaces, for example when the WiFi link is lost and only the 3G/4G link remains, the connection is not interrupted but automatically switched on the remaining link, increasing the robustness of the connection experienced by the user. The behavior is the same in the opposite situation where a new link layer connection is available, splitting the traffic between the two channels and increasing the throughput.

Then, multipath transport protocols have several theoretical benefits on the performance of a connection. We said that the realization of these objectives is not straightforward because usually we have to handle with multiple links with different characteristics. A well-known problem of multipath connections is called *head-of-line blocking*, that arises in presence of links with high differences in bandwidth and delay. If we want to guarantee in-order delivery of the packets at the receiver, results that the packets that are scheduled on the *low delay/high bandwidth* link have to wait for the arrival of the packets from the *high delay/low bandwidth* link in a temporary buffer before being passed to the higher layer protocol. *Head-of-line blocking* problem causes burstiness in the stream of the data due to the delayed delivery of the data to the application layer. In this scenario application becomes less reactive and user experience degrades becoming worse than with a traditional single path connection. In case of a limited buffer of the receiver this problem could cause also dropping of some collected packets further

degrading the quality of the connection, then a well designed buffer at the receiver is a good way to tackle with *head-of-line blocking*. In [33] and [3] is recommended a buffer size of

$$\text{Buffer} = (\sum_{i=1}^{n} \text{bw}_i) \times \text{RTT}_{\max} \times 2 \tag{2.11}$$

where $\sum_{i=1}^{n} \text{bw}_i$ is the sum of thee bandwidth of the n links and $\text{RTT}_{\max}$ is the highest Round Trip Time (RTT) among all the subflows, allowing all the subflows to transmit at full speed also in case of erasures.
Regardless the solution to the problem of multipath transmission given by MPTCP, the core of the protocol at the transmitter side is the *scheduler*. The scheduler is the entity that decides how to split the traffic between the available links and a wrong scheduling decision might result in the problems that we have previously explained. In [3], the authors have simulated a network with a transmitter and a receiver that communicate through two subflows as in Fig. 2.10, using MPTCP.

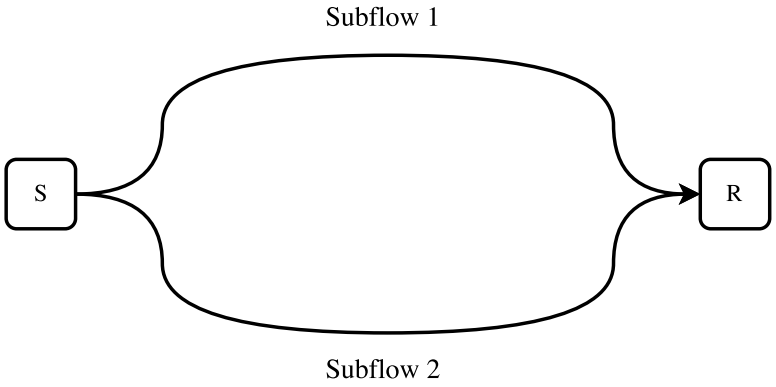


Figure 2.10: Testbed of [3]

With this simple setting they have tested four different schedulers from the simplest one, called Round Robin (RR) where packets are transmitted on a path or on the other in alternation, to complex ones that take into account estimated parameters of the links as the RTT or the congestion window of the TCP socket at the receiver, reaching the conclusions that designing a scheduler compatible with the congestion control of TCP, that can tackle the buffer limitation at the receiver and the *head-of-line blocking* at the receiver is not trivial especially with large RTT differences. In addition, the design requires very precise estimation of the RTT that is also an hard task in highly variable scenario like wireless

networks.

2.4.1 Network coding based MPTCP

In [4], the authors tried to give a solution to the common problem of *head-of-line blocking* that affects all the MultiPath protocols, proposing a NC-MPTCP protocol. This protocol mixes network coded and normal flows in order to compensate for the lost or delayed head-of-line packets but also includes a scheduler and a redundancy estimation algorithm.

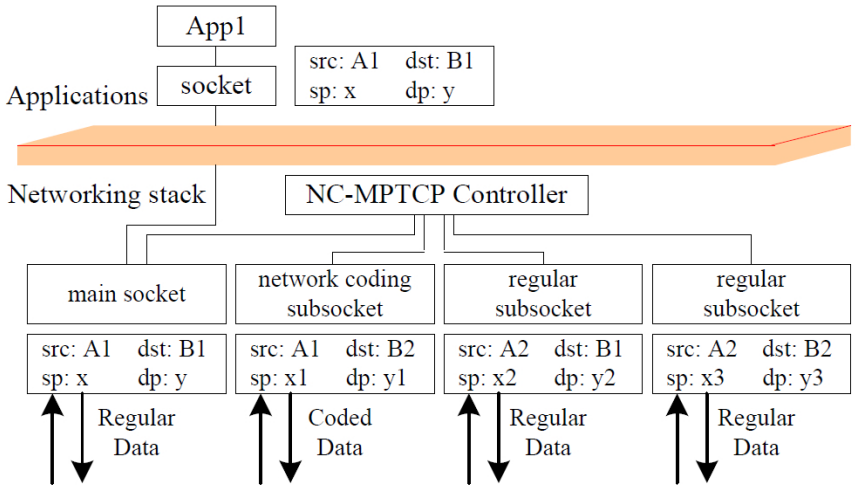


Figure 2.11: Protocol architecture presented inn [4] for NC-MPTCP.

In Fig. 2.11 is possible to see the architecture presented for NC-MPTCP that consists of several modules: the *main socket* that is a normal socket that acts as a medium between the application layer and the NC-MPTCP controller, the *NC-MPTCP controller* that implements the scheduling algorithm and the redundancy estimation. The *regular subsockets* and the *network coded subsockets* are standard TCP sockets managed from the main socket that transmit data with regular TCP and network coded TCP, respectively.
The scheduling algorithm in this is thought in a reverse way, instead of choosing the path on which to send the packet, it waits for an available subsocket and then chooses the best packet to send among those in the queue. The network coded subflow is utilized only as redundancy flow in order to compensate losses and delay on the regular TCP flows, the quantity of packets transmitted on the network

coded flow is computed with an expression that is function of the RTT and the probability of losses of the links and have to satisfy the following conditions

$$|C|(1-p_n)+|G'|(1-p_r)\geq|G| \tag{2.12}$$

$$|C|\geq|C|p_n+|G'|p_r \tag{2.13}$$

where p_r and p_r are the probability of losses on the regular and network coded flows, G denote a *generation* and C the set of coded packets computed over G; network coded subflows transmits C and the regular ones transmits G' such that $G' \subseteq G$. The expression used for the redundancy estimation try to find the minimum $|C|$ and $|G'|$ that satisfy equations (2.12) and (2.13). [4] found out via simulation that NC-MPTCP requires a smaller aggregation buffer at the receiver than MPTCP in order to achieve a stable goodput and in general is more robust to high differences of delay of the subflows, which is the real weakness of MPTCP. Another example of the application of Network Coding to MultiPath TCP is reported in [5], where a full network coded protocol instead of a partial one as the in [4] where the network coding subflows was used only for the redundancy packets. The scheme used is reported in Fig. 2.12 where we can see that all the packets pass through two steps of Network Coding and the second layer of encoding provides the redundancy packets to overcome the packet losses on the link.

The authors of [5] have computed the end-to-end analytical throughput of their protocol, providing a closed form expression, function of the parameters of the links. Then they have applied to this analytic model and the classical MPTCP the data from three heterogeneous links (WiMax, WiFi mesh network and an Iridium satellite link) collected by a moving vehicle in order to have a dataset that reflects the real utilization of the devices. The results from these experiments show that the classical MPTCP is hindered by high packets losses while the developed version of NC-MPTCP reaches a higher throughput because is able to mask losses to the TCP sockets. Another benefit of using a fully network coded protocol is that it simplifies the scheduling and the retransmission protocol. The *scheduler* is simpler because all the packets from the same generation have the same characteristics, and without the need to choose the optimal packet to send on

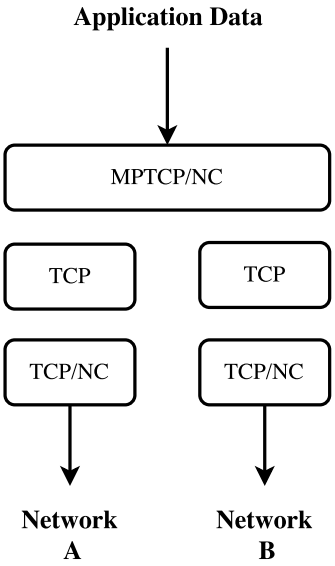


Figure 2.12: MPTCP with NC in [5].

the free channel. Second, the *retransmission protocol* has an easier management because we only need to acknowledge the number of *degrees of freedom* at the receiver and retransmit a sufficient number of linear combinations; if a packet is lost, any packet sent in a different subflow can be used in the lost packet's place. In order to prevent the dumb retransmission of packets in this case as in [2] for each *degree of freedom* to send $R \geq \frac{1}{1-\epsilon}$ packets are transmitted where ϵ is the probability of erasure of the link. The examples presented from [4] and [5] show how is possible to significantly increase the quality of service in a multipath scenario by smartly exploiting Network Coding, and how this technique is able to tackle losses on the links and the emergence of situations as the head-of-line blocking. In addition, it allows to reduce the memory requirements at the receiver and simplifies the scheduling and feedback protocols that are not trivial when there are multiple links to manage.

2.5 Simulators

New protocols and network technologies are constantly under research and development in order to provide every time better performance with respect to the state of the art. The most complete method that can be utilized to evaluate the

performance of a protocol is the *real implementation* of a sample system. A real implementation returns the exact results that the protocol will have when utilized in a real world setting because all the parts and modules of which is composed are running on physical devices, in particular in network applications, the channel is real and not simulated. This approach encounters also some difficulties, firstly the cost because network protocols have to run on a network with several devices that sometimes have prohibitive costs; an example on the theme of our work may be the testing of a multipath scenario with a smartphone that communicate via WiFi and LTE; in order to have a LTE link we need a LTE base station with a dedicated frequency to use for tests, that is an expensive machine with a cost that can reach 100K euros. Second, it is not always possible to reproduce all the interesting scenarios on which the protocol could run because every time we have to reconfigure the devices with different parameters or try to reproduce different channel conditions. Finally, it is also *effort and time expansive* because it requires to program the devices with low level programming language, interfacing with problems that could arise as compatibility with operating systems and firmwares. For these reasons the real implementation of a new system usually is one of the last steps in the development of a network protocol, after a phase of testing by software that can prove the effective benefits and relevance of the results and justify the investment for the implementation of a testbed.

The software technologies utilized for these evaluations can be divided in two families that are differentiated by the approach utilized to reproduce the entities in the network, *emulators* and *simulators*. The family of the *emulators* is composed by the software tools that are halfway between testbeds and simulators. An emulator allows the developer to create a simulated network of devices that interact with each other them through the standard network protocols or with custom developed protocols. Then with an emulator it is possible to choose the topology of the network and the typology of the link between the nodes. These nodes can run a chosen protocol stack allowing the developer to evaluate the behaviour of its project when inserted in a network. In addition various channel scenarios can be simulated, for example the interference between wireless devices or the variability of the noise of the channel due to external events. These are only a subset of the opportunity brought by an emulator; some examples of famous network emulators are Network Simulator (NS) [34] and `mininet` [35]. The

first allows to simulate the *network interaction* between devices providing models for all the most used protocols from physical to application layers, technologies as WiFi and LTE, routing protocols. The latter, `mininet`, provides to the developer the tools to *emulate full devices* and not only the network interactions between them. `mininet` allows to run real kernel, switch and application code on simulated devices; connected devices in `mininet` are real Vitual Machines (VMs) that the developer can control from a simulated command line. NS is then a *network emulator*, that aims to allow simulated components to communicate with protocols that are commonly used in the real world; `mininet` instead is a *environment emulator* that extends further the previous model in an attempt to build an implementation environment in which real world protocols may be directly executed, simulating the environment of a particular operating system.
Sometimes the evaluation of a protocol or an algorithm is not strictly or immediately required in relation with real-world connections, because it is not clear how a new technology can be merged with existing networks or, conversely, be applied to so many scenarios and then it is more useful to see the performance abstracting the underlying network. We have seen in the previous sections examples about Network Coding that show the gains of this coding scheme that are not dependant to a particular transmission protocol but show benefits with respect to the usual *store and forward*. The examples presented are designed to be easy to understand graphically and show immediately the results in the most simple possible setting. If we want to go deeper in the understanding of which are the possible benefits in different scenarios for example with an higher finite field size, different topologies and characteristics of the transmission we have to write our mathematical model in a programming language and compute the results with the imposed parameters. This is what is called a *simulator*, because it simulates the behaviour of a system through the numerical computation of the algorithms and mathematical expressions that describe it. This approach abstracts all the systems that in real world work at lower layer as contained in a black box, managing only the aspects that are included in the mathematical model. Thus, a *simulator* provide to the user only a partial view of what may be the developed system in a real environment, focusing only on the interesting aspects. Inserting a new developed system in a complex protocol stack maybe misleading for the understanding of the behaviour of the new product, because some situations

could arise not as an effect of our work but for reasons that regard the underlying protocols; then a simulator is useful in order to focus the attention on a subset of parameters and characteristics that interest us.
A particular case in the family of simulators are the *discrete event simulators* [36] that differ from the others in the way time is simulated. In this type of simulators the time is not divided in equally spaced intervals during which the behavior of the system is tracked. The timeline in this typology of simulators is broken by the *events*, then the length of the time slices can vary in relation to the happening instant of the event. The type of events that interrupt a time slice is decided by the model, and has to be a particular event that changes the state of the system. This feature of not to take into account every regular unit of time make usually the *discrete event simulators* faster in the simulation with respect to other types of simulators [37].
For these reasons in our work we have chosen to employ a *discrete time simulator*, in order to focus on the results of the developed protocol and not on the interaction with the lower layers, abstracting the links that connect the node and its characteristics in the simulated network, and in order to have relatively good simulation speed.

3
Contribution

We have seen that Network Coding brings several benefits in terms of throughput in various scenarios as multicast and broadcast communication, and with inter-session network coding also in the case of multiple unicast flows that share the same network. In case of lossy links, we have seen that NC increases the robustness of the connection and furthermore simplifies the managing of the retransmission scheme abstracting the role of the sequence number in the ARQ scheme. A scenario that takes particular advantage of these benefits in lossy networks is the *multipath case*, that without NC has several complications as discussed before.

The leading thread between lossy networks and the benefits that NC brings in that situation are the *feedbacks*. Feedbacks are the only medium in a packet network utilized for meta-data transmission from the destination to the source about the connection and the only way to smartly exploit the Network Coding features. In fact, feedbacks act as a *control knob* of the coding procedure at the source that set the number of packets that the source has to generate as a function of the receiver's state and the channel measured state. A wrong feedback management can completely nullify the gain of the coding scheme potentially overwhelming the network of useless packets or even, get worse throughput, when few packets are sent that do not allow the destination to decode the full information in a useful time. Thus, a reliable feedback procedure is of central importance to

guarantee that the transmission can reach the aforementioned gains of this novel coding scheme but also that the network is utilized in a fair way and not flooded of useless packets. Some examples of the large gains in multipath scenario (in particular MultiPath TCP) that a smart feedback management can provide are shown in the previous section and are referred to the researches in [4] and [5]. In these researches but also in those presented before like [14] the ARQ scheme is combined with Network Coding scheme to provide the best possible performance and reliability. Every time that feedbacks are analyzed, they are reported as a perfect stream of meta information in the sense that the packets that carry the feedback information are transmitted through a perfect channel with infinite bandwidth, without delay and errors. This assumption is obviously optimistic and never applicable in a real network and then also the results are spoiled by that. The first step of our work is then the research of what are the effects of a non-perfect feedback channel in both single path and multi path scenarios.
In order to do this we have built what we have previously defined as a *discrete event simulator* that measures the performance of a transmission in these two scenarios and in different setup of the links. All the transmissions that we have analyzed utilize Network Coding, in particular, Random Linear Network Coding; thus, we need some tools that allow us to easily manage all the phases of a network coded transmission. The set of tools that we have utilized is a library named Kodo [1] that we will describe in the next section.

3.1 Kodo

Kodo is a project developed by the Steinwurf team initially composed by students of the Aalborg University and then expanded with other members from different academical organizations. It was born as a C++ library for Network Coding that aims to help students and researchers to develop network coding algorithms. It provides several ready-to-use building blocks and algorithms that can be exploited by the developer to simplify and speed up the process of development of their project. Kodo is then a flexible and extensible Application Programming Interface (API) that provides already functional components to reuse. The first objective of this project is to be easy for users with any level of programming experience, hiding the more complicated implementation details. The ease of

use should no interfere with the performance that is one of the key challenges of Kodo; if this is not possible, the tradeoff prefers always the ease of use, in fact in the case that an high performance implementation is required it is possible to modify directly the source code of Kodo that is open.

Figure 3.1: Kodo logo.

The main part of the library is dedicated to support the creation of Random Linear Network Coding (RLNC) systems, then provides the tools to manage the parameters that we have illustrated in the introduction to Network Coding. First it is possible to create the fundamental entities of the *encoder*, that generates the linear combinations of packets, and the *decoder*, that collects the incoming packets until it has received a sufficient number of linear combinations and is ready to decode. The data to be encoded can originate from a finite or an infinite stream, but in order to make feasible the encoding operation, the transmitter divides the stream in chunks of a reasonable size. A chunk is what we have previously defined as a *generation* and when a couple encoder/decoder is created is possible to set the *generation size* K, i.e., the number of packets and the *symbol size* d, i.e. the dimension in bytes of each packet in the generation.

In Fig. 3.2 an example is reported of the basic algorithm implemented in Kodo to partition a stream of N packets in $M = \frac{N}{K}$ generations, each containing K packets. The third parameter that Kodo allows to customize in an encoder(decoder) is the field size q. This parameter set the dimensions of the Galois Field on which to perform the coding operations, modifying the set of allowed coefficients used to computed the linear combinations of packets and translating all the packets from a binary representation to a finite field of size q representation. The choice of the finite field size q is a key parameter for a simulator, because there is a tradeoff between the speed of the simulation and the coding performance; anyway, it could be interesting to test the performance in different field, this choice

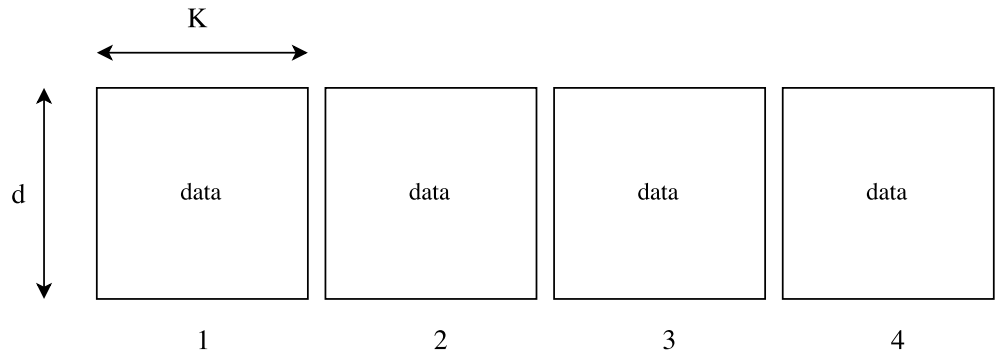


Figure 3.2: Basic partitioning algorithm.

has a different relevance than if it was implemented in a real network where the complexity constraint of the devices has a decisive importance on the applicability of the employed code. The fact that these functionalities are ready-to-use, is very helpful for a developer, that does not need to build an environment for the finite field computation; this might be annoying and a waste of time that could be employed in the research objectives. In this context, Steinwurf team provides another tool that is specific for computation in finite field arithmetic named *Fifi* [38] particularly useful in contexts as erasure correcting codes and cryptography. The benefits of using Kodo are not confined to these few basic features for RLNC, but consists of others important functions that allow to exploit the potentiality of this coding scheme. The feature that makes Network Coding unique with respect to the other codes is the possibility to *recode* packets at intermediate nodes, generating other linear combinations starting from the collected packets. This is implemented in the function `recode` of Kodo and easily employable in a project. Other functions concern the different encoding strategies that are possible to employ at the transmitter to tackle some problems that can be critical in some settings as the *decoding delay*, some of those we have described in the introductory part. One example could be the *systematic encoding* where not all the packets are transmitted as linear combinations but a part of the traffic consists of uncoded packets and the remaining part of coded packets; this technique is analysed in [25] where are compared the performance with the standard approach. There are scenarios where the data at the transmitter are not fully available at the same time but becomes available at different times, and potentially at variable rates; in these cases there could not be enough packets to complete a generation at

the encoder and we want to send packets with the temporary available subset. This technique of encoding is implemented in Kodo under the name of *on the fly encoding*. To reduce the *decoding delay* in Kodo *Sliding Window* encoding has been introduced, described in the section 2.3; the linear combinations are not generated from the whole set of packets of the generation but only from a window that shifts on the generation when some packets are acknowledged.
These are only some of the functions implemented in Kodo to help researcher in the network coding field, for example there exist several schemes with a different strategy on the random selection of the coding coefficients that may impact on the code performance. In order to increase the flexibility of the library, the developers have exported some of the functionalities of Kodo in other programming languages such as C or Python. We have decided to exploit the benefits of a high level language as Python and then we have employed the *Kodo-python* library in our simulator.

3.2 Simulator

In the previous paragraph, we have seen how Kodo can be a useful tool for the development of algorithms and protocols that employ Network Coding as transmission paradigm. Due to its flexibility, this library can be embedded in every type of implementation like a simulator, an emulator or a real testbed. Within the collaboration with the Technische Universität Dresden (TUD) a protocol was developed for multipath transmission, as those of the family of network coding based Multipath TCP. We call this protocol `nctun` meaning *Network coding tunnelling*. Its main idea is reported in Fig. 3.3.

The principle of a multicast connection is to merge the potentiality of multiple available links to send the information from the same session,that is not possible with standard protocols (as TCP) that use no more than one link for each session. Upon reception of a request to open a new TCP session the operative system of a device chooses one and only one of the available layer 2 links. The protocol consists of three important steps:

1. The first step of this protocol is the creation of a *virtual interface* that masks the multipath behavior of the protocol to the operative system, represented

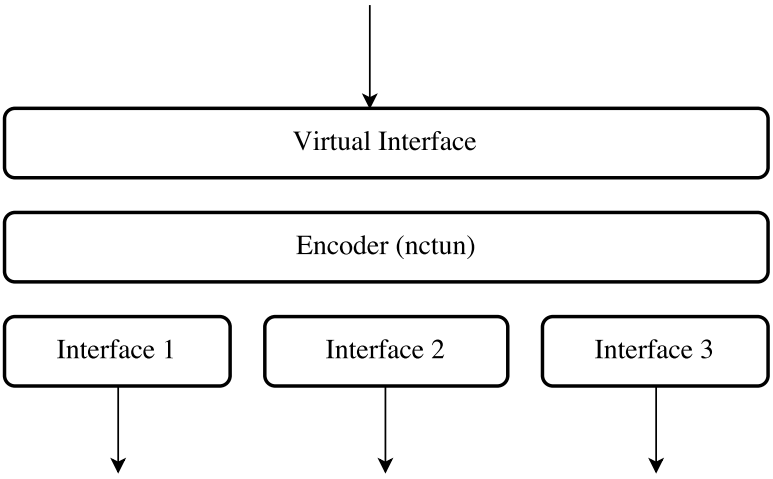


Figure 3.3: nctun protocol.

in the first rectangle in Fig. 3.3. This trick does not hide the standard links to the operative system but adds a new layer 2 interface that is *virtual*, then the system can choose among a standard single path connection or a multipath one to transmit its information.

2. The second part is the *encoding* of the information, that is applied the whole stream that is passed through the virtual interface. In this way we can exploit all the benefits of a full network coded transmission in terms of throughput, robustness and retransmission management. In other examples of NC-MPTCP other encoding strategies have been employed, as in [4] where only a part of the information sent is network coded.

3. The third and last step is the *scheduler* that aims to choose at each time the best path to send a packet onto. As we have seen, it is fundamental in MPTCP in order to achieve the best performance but its importance has been reduced with the introduction of the mixing properties of Random Linear Network Coding [4] [5].

Once selected the path to follow, the transmission continues as a normal transmission in a TCP network.
At the receiver's side there must be another complementary entity that is able to collect the different flows and pass them to the decoder in order to retrieve the

original information.
`nctun` is a system that promises very high gains in all the aspects with respect to the usual single path connection. The key problem with this new protocol is the implementation in a real system due to the need to create the virtual interface, which has non trivial compatibility issues with the operating system. In addition, as said in the section 2.5, in order to have some reliable results about the performance we have to arrange a multipath network, for example a WiFi with an LTE dedicated channel to reproduce a smartphone use case, that are a complex requisite to satisfy due to the prohibitive costs of prototyping.
For these reasons, we have decided to accompany the implementation of the protocol with a simulator that would abstract the transmission links and the protocol stack and would show the performance of the developed multipath protocol in general conditions. The whole project has been developed with the Python programming language [39], that is a high level interpreted object-oriented language. It has the benefits of being very flexible and light, but with an enormous standard library that provides tools for any task.

Figure 3.4: Python programming language logo.

In addition, its syntax is simple and emphasizes the code readability and is supported by several developers that provide useful tools not included in the standard library; an example of this support by external communities is the `Kodo-python` library [40] that we have employed in our project and that allows Python to be more and more flexible.
The core of the simulator replicates the transmission of network coded packets from a transmitter to a receiver through several links, an example with two paths is reported in Fig. 3.5. Each simulated link has in addition a *reverse path* that is used to transmit informations in the reverse direction, in our case the feedbacks about the transmission. In Fig. 3.5 these *reverse paths* are represented with the green dashed lines.

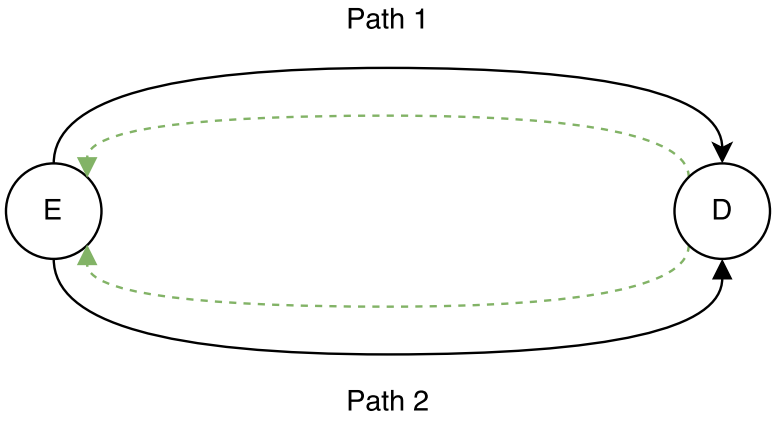


Figure 3.5: Topology of the simulated multipath network, the black solid lines are the *forward path*, the green dashed lines represents the *reverse paths*.

The transmission of the information proceeds in a way similar to the `nctun` protocol presented before:

1. The information from the upper layer is passed to the *transmitter*
2. The transmitter *encodes* the information dividing the stream in several *generations* of K packets. From each generation$n$ encoded packets are created , where n depends on different parameters of the protocol and the links.
3. The encoded packets are passed to the *scheduler* that decides on which link to transmit the current packet.
4. The *propagation* of the packets is simulated, with different parameters for each path.
5. After the propagation time the receiver *collects* the packets incoming from the different paths.
6. When the receiver has collected enough packets, it decodes the original information and pass it to the application layer.

This is a brief outline of the basic procedure that has been implemented in every version of the simulator.
The *first point* of the simulation is the injection of the source data in the transmitter. In a simulator it is not important the exact data that we transfer over

the network because our focus is on the performance of the transmission, and the nature of the source data does not have any effect on that. What we want to control are the *offered traffic* at the encoder and the dimension of the source chunk of data that in the code is referred to as `block_size`. For these reasons, each time that is needed, a chunk of `block_size` bits dimension is randomly generated, using the `os.random` function provided by the standard library of Python, as reported in Fig. 3.6.

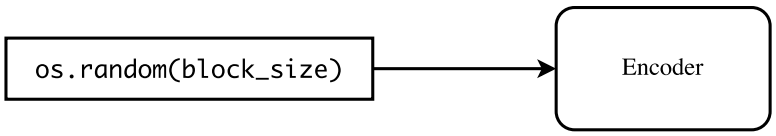


Figure 3.6: First step.

In the *second step* the Random Linear Network Coding scheme is applied to the chunk of bits that arrives from the source. The *Kodo-python* library provides a tool called *encoder/decoder factory* that is an entity that generates encoders (or decoders) with the same parameters. We can associate with each encoder a block of bytes to encode, in other words, the encoder performs the role of a single generation. The encoder performs the aforementioned *partitioning algorithm* that divides the block of data in packets of a determined dimension `symbol_size`, where packets can be at most `n_symbols`. In addition Kodo allows to choose among different `encoder_factory` functions, in order to choose the right size of the finite field in which to perform the encoding; in this case, the Python version of Kodo shows some limitations due to the performance constraints of the Python language in the field of pure mathematics computation with respect to C++ that allows to set larger field sizes. Once assigned a block of data to the encoder, we can generate as many encoded packets as we want to call every time the encoder through the function `write_payload` that creates a encoded set of bits that we can use to simulate the transmission in our system. Usually in our system we use the encoding strategy called *Sliding Window*, that creates coded packets from linear combinations from a subset of symbols of the generation. The number n of generated packets is computed differently with respect to the particular version of the feedback protocol that is simulated.

Each generated packet is sent on one of the available links, the procedure of selection among the channels is called *scheduling*. The basic scheduler that

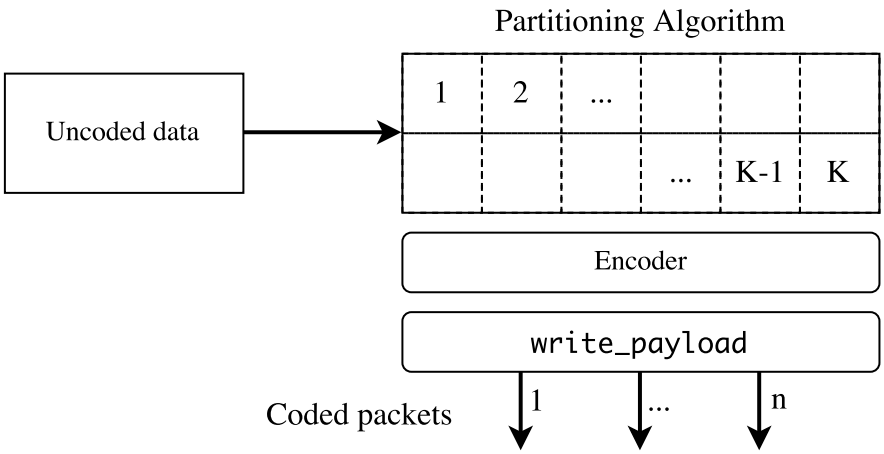


Figure 3.7: Second step: *encoding*.

we use try to fully exploit the available bandwidth at each link, sending the current packet on the first free link. This scheduler achieves optimal performance assuming that the sender knows immediately when the channel is available for the transmission of a new packet. In order to do this, we maintain a different timer t_l for each link l that is updated at every transmission (for this reason this is a discrete event simulation) adding the time necessary for the transmission $t_{\text{pck},l}$ of the packet in the l-th link; e.g., for the k-th transmission

$$t_{l,k} = t_{l,k-1} + t_{\text{pck},l} \tag{3.1}$$

In this way we can select the link $\hat{l}$ that has the minimum value of the simulated time t_l in that moment

$$\hat{l}_{k+1} = \arg\min_{l} t_{l,k} \tag{3.2}$$

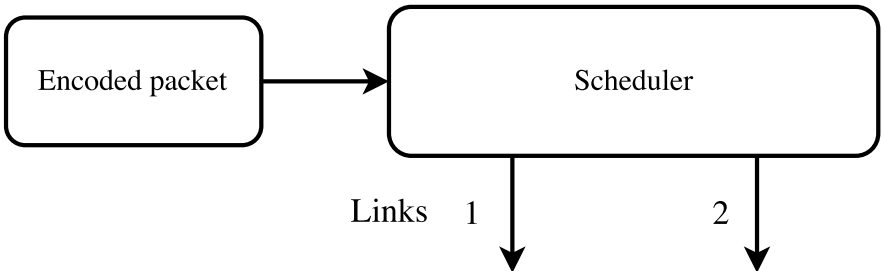


Figure 3.8: Third step: *scheduling*.

Other schedulers have been tested in addition to this one, for example a scheduler that works with the assumption of complete information about the channel state in each precise moment, i.e. bandwidth and the latency; in this way the system is able to choose the path that, in case of successful transmission, will first deliver the packet. In the simulation runs we have always used the first presented scheduler.
In the *fourth step* the *propagation* of the packets along the links is simulated, that concerns in the simulation of *bandwidth*, *delay*, and *loss rate* of the selected link. Each time that a packet is scheduled for a link, it is decided if it will be erased during the transmission or will be successfully delivered, following a Bernoulli distribution $\mathcal{B}(p_l)$ where p_l is the actual value of the loss rate of the l-th link. If the transmission is successful the packet is inserted in a queue, with attached the time of reception calculated as

$$t_{rec} = t_l + t_{\text{pck},l} + d_l \tag{3.3}$$

where t_l is the instant measured by the timer of the l-th link at the moment immediately before the transmission, $t_{\text{pck},l}$ is the time necessary for the transmission of the packet over the l-th link and d_l is the delay associated to the link l. The time required for the transmission $t_{\text{pck},l}$ is function of the dimension of the packet `dim` and the bandwidth of the selected link b_l

$$t_{\text{pck},l} = \frac{8 \cdot 10^3 \cdot \texttt{dim}}{10^6 \cdot b_l} \tag{3.4}$$

where $t_{\text{pck},l}$ is expressed in [s], `dim` in [bytes] and b_l in [Mb/s]. The parameters of *bandwidth, latency* and *loss rate* can be set by the user and can change dynamically during the simulation. When a packet is elected to be erased during the transmission, it is not inserted in the queue but the timer is updated in the same way because we have to simulated a correct injection of the packet in the channel with a failed propagation along the channel, then we have to simulate the flowing of time also for this event.

In the *fifth step*, the receiver listens for the incoming packets from the multiple links and forwards all of them to the decoder. In addition to the individual timers of the paths that have to track the different speeds of the time with respect to the

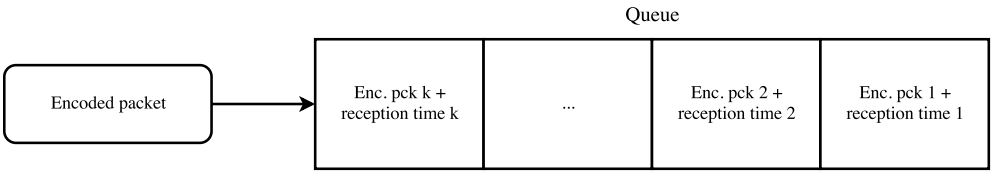


Figure 3.9: Fourth step: *propagation*.

events (i.e., the transmissions), there is another timer that is *system time* T that represent the time reference for the entire system. The *system time* is updated after each transmission in according to the individual timer in order to have a coherent simulation of the time

$$T = \max_{l} t_l \tag{3.5}$$

If $t_{rec_{k,l}}$ is the reception time of the k-th packet in the l-th link queue, when there exist k, l such that

$$t_{rec_{k,l}} \leq T \tag{3.6}$$

the associated packets are considered as *delivered* at the receiver.

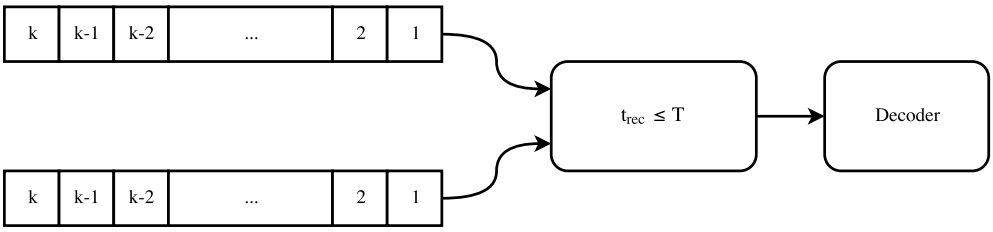


Figure 3.10: Fifth step.

The sixth and last step of the core of the simulator is the *decoding part*. In this part the received packets are passed to the decoder the performs the decoding procedure. Every time that a packet is delivered we use the Kodo function `decoder.read_payload` that extracts and saves the payload of a packet and adds its coefficients to the decoding matrix. This function automatically performs Gaussian elimination [24] on the matrix of the received coefficients in to order update the rank of the matrix. If the rank is augmented with respect to the previous value, it means that the new packet has brought what we have previously defined as one *degree of freedom*. When the decoder has received

enough *degrees of freedom*, the function `decoder.is_complete` will provide a `true` boolean value; then is possible to retrieve the original information using the function `decoder.copy_from_symbols`.

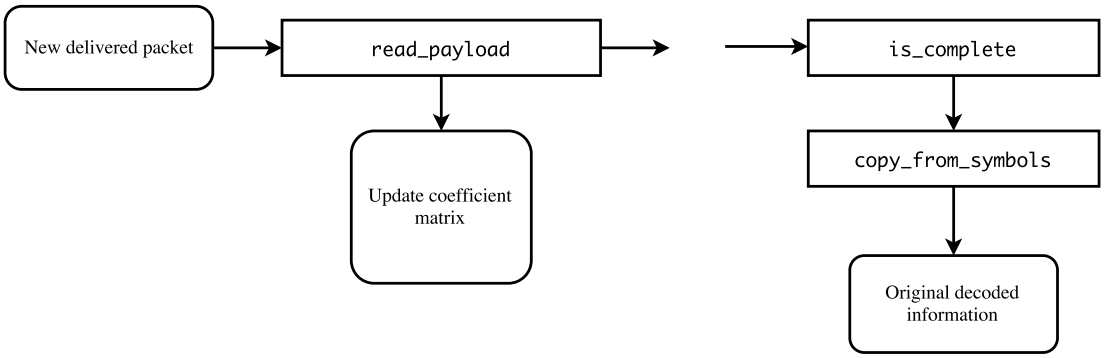


Figure 3.11: Sixth step.

It is important to note that the described procedure is the central functioning of the developed simulator, the start point on which are based the performance evaluations of different feedback management algorithms that will be presented in the following section. At the beginning of the simulation is possible to set all the parameters of the network from the command line:

- *Bandwidth* of links, chosen independently in `Mb/s`
- *Latency* of links, chosen independently in `ms`
- *Loss rate* of links, chosen independently in `%`
- *Number of symbols* of the generations
- *Size of symbols* in `Bytes`
- *Number of generations* to transmit

It is possible to set other parameters that are related to the feedback algorithm and then we will describe them later. In the following section we will see how this simulator has been employed to analyze different strategies for the management of the feedbacks in these network settings. Our objective is to search some results about the behavior of the whole system when the feedback are transmitted in different conditions, deleting the assumption of perfect feedbacks that is

usually employed in related works. In addition, we will try to exploit from these results some rules about how to send and manage the feedbacks and the related retransmissions.

3.3 Analyzed protocols

In the sections 2.3 and 2.4 we have seen the solutions provided by the literature [14] [15] for the implementation of a feedback structure in a Network Coding system. In this type of systems the feedbacks have two main functions [14]:

- Notify the packet erasures and then control the necessary retransmissions
- Optimize the encoding procedure during the transmission process

the first is the classical function of the ARQ schemes, with the proper modifications in the case of network coded transmission (see section 2.3), the second is possible in the case that we use particular Random Linear Network Coding versions as the Sliding Window also described in section 2.3. This informations have to be included in the feedback packet that will be sent in the *reverse channel*, one of the two green paths in Fig. 3.5.

Figure 3.12: Feedback packet scheme.

The feedback packets have to transport a little amount of informations in order to be transmitted as fast as possible; thus, we just include two fields. In Fig. 3.12 is reported the structure *feedback packet* that we have used in our simulator:

- The first field is the *last seen symbol/packet* k that is related to the Sliding Window paradigm that does not encode the already *seen packets*, but only the part that has not been seen or decoded, decreasing decoding computation complexity and the decoding delay. If the index of the last seen packet is k, and the encoder knows that, the encoding window will start from the $(k+1)$-th packet.

- The second field brings the information about the number of *degrees of freedom* effectively received. This information is useful in order to estimate the number of encoded packets that the transmitter has to generate. For example, in case of lost feedbacks the encoder can rely on the last feedback received and transmit a number of encoded packet that will probably allow decoding.

We have developed three protocols that manages in different ways the feedback transmission and the retransmission of other packets from the sender.

3.3.1 Full-duplex protocol

The first protocol that we have developed is thought for network that have links capable of transmit packets and receive feedbacks in the same time, that for simplicity we will call *full duplex* links. The aim of this protocol is to fully exploit the available links' bandwidth, and the scheme is presented in Fig. 3.13.

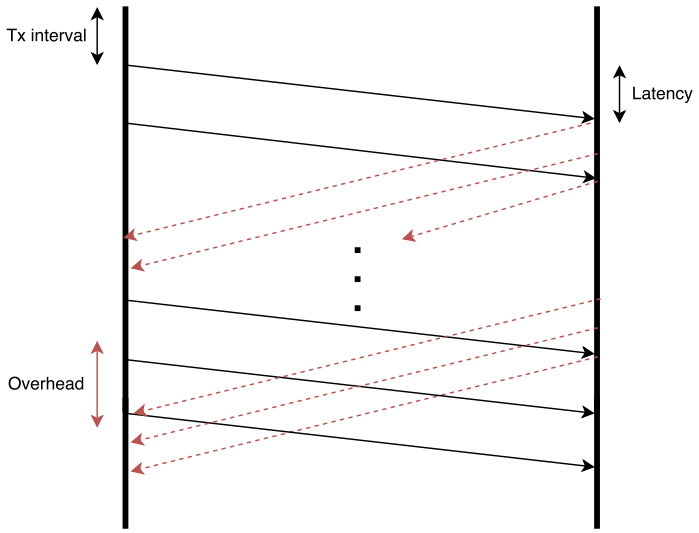


Figure 3.13: Scheme of the first protocol, black solid arrows represent the encoded packets, red dashed arrows represents the feedback packets.

The *full-duplex* protocol consists of by the following steps:

- The sender fills all the available interfaces with the encoded packets, generated following the Sliding window paradigm

- For each packet received, the decoder sends a feedback composed as previously described
- The sender stops to generate new packets when it receives the feedback that confirms that the receiver has successfully decoded the original information
- In case that we want to send *multiple generations*, the second generation starts to be transmitted after the first is completely decoded.

In Fig. 3.13 we can see on the left side the transmitter and the black solid arrows that represent the outgoing packets, while on the right side we have the receiver that sends feedbacks at each reception, the red dashed lines. We can immediately note how in the figure are reported some feedbacks between two received packets. This is due to the way that we send the feedbacks, that is the core arguments of the research. In the *multipath scenario* that we consider, we transmit the *feedback packets* only on one of the two available *reverse links*. Feedbacks are small packets that we want delivered reliably as soon as possible, there are no reason to use two links in order to transmit such a small packets incurring in the aforementioned managing complications of multipath connections.
Thus, the scheme presented in Fig. 3.13 is the one that reports the functioning of the protocol on the path selected to transmit the feedbacks, and the red arrows between two received packets want to represent the transmitted feedbacks due to the reception of packets from the other path. They also want to underline that the two links have different characteristics and that erasures are random, then the packets that travels in the two paths are not synchronized in any way. Thus the other path will only transmit the packets in the *forward direction*, and feedback task will be left to the selected path, that we will refer as the *feedback path*.
Still in Fig. 3.13 on the lower left is reported a red arrow that underline the *overhead* packets, i.e. the packets that are transmitted but are not useful because the receiver has already received enough packets to decode the original information. This happens due to the delay of the delivery of the last feedback packet, during this time the sender will transmit several packets that are useless and will be dropped by the receiver. The higher the delay and the bandwidth, the higher also the *overhead*.
In order to reduce the number of transmitted feedbacks we have tested also another version of the same protocol, where we transmit feedbacks for all the packets

of the same generation received when the receiver is able to decode, the scheme is reported in Fig. 3.14.

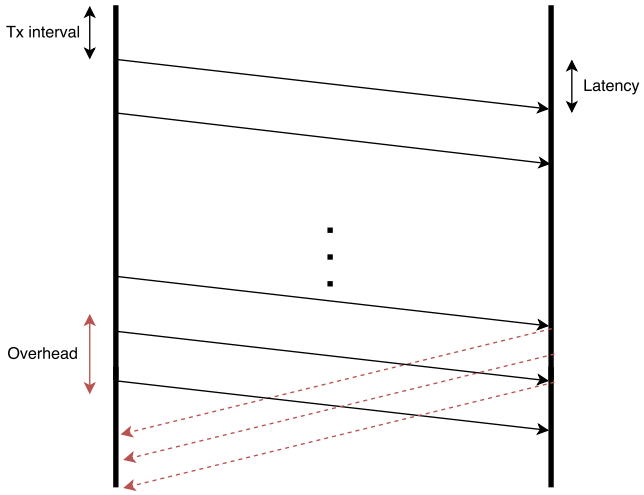


Figure 3.14: Scheme of the second version of the first protocol.

Unfortunately this protocol does not provide to the encoder the informations that allows to exploit the Sliding Window, but significantly reduces the number of transmitted feedbacks.

3.3.2 Half-duplex protocol

After the first basic protocol that we have developed for *full duplex* channels we have thought of a protocol that would be compatible with links that cannot transmit and feedbacks in both the directions in the same time, like the *half duplex channels*.

The protocol tries to solve the problem of unidirectionality of the links sending block of n encoded packets that will be acknowledged at the end of the transmission. Formally, the protocol follows these steps:

- The transmitter generates n encoded packets and forwards them in a row without interruptions. The quantity n is the result of the expression

$$n = (K - \text{rank}(D))(1 + r) \tag{3.7}$$

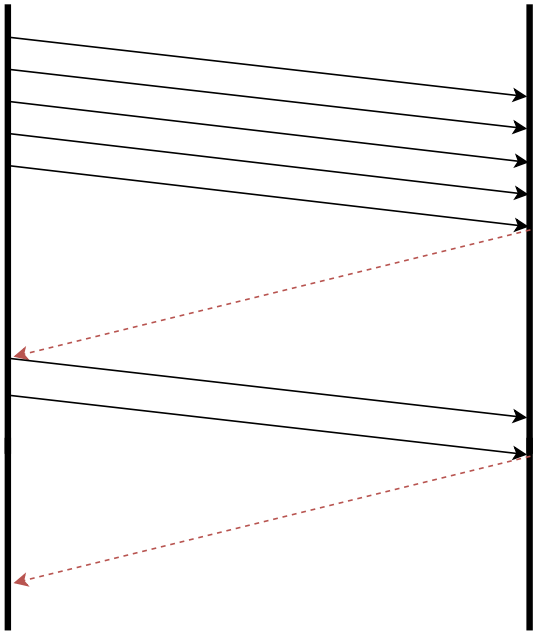


Figure 3.15: Scheme of the *full-duplex* protocol, black solid arrows represent the encoded packets, the red dashed arrows represents the feedback packets.

where K is the number of symbols of the current generations, D is the decoding matrix, rank(D) is the number of *degrees of freedom* received at the decoder and r is a *redundancy* factor, with $0 \leq r \leq 1$. In the first iteration rank(D) $= 0$.

- At the end of the block transmission the decoder sends f feedback packets, composed as described in the introduction of Section 3.3. The number of feedbacks is in general calculated in order to provide a sufficient robustness to erasures; given the loss rate ε of the *feedback path* and assuming i.i.d. losses the default number f of transmitted feedbacks is

$$f = \frac{1}{1-\varepsilon} \quad \text{for } 0 \leq \varepsilon \leq 1 \tag{3.8}$$

- If the decoding is completed, finish and pass to the next generation; otherwise update the rank of the decoding matrix known at the transmitter with the new information acquired with the feedbacks and return to the first point.

We call *round* each transmission block with the related feedbacks. In this protocol, there is a throughput decrease during the transmission of the feedbacks and then the optimal case is when we complete the transmission after the first round, without requiring another retransmission round. The redundancy r is a parameter set at the beginning of the simulation and in order to have a good robustness to losses in the literature is suggested to use r

$$r \geq \frac{1}{1 - \varepsilon_{\max}} \tag{3.9}$$

where $\varepsilon_{\max}$ is the maximum among the loss rates ε_l of the available links l. The number of rounds required to terminate the transmission directly depends on the *loss rate* of the paths and the *redundancy* r.

In this case the Sliding window protocol is affected by the lack of feedbacks during the transmission but in case of retransmissions it can exploit the information carried by the feedbacks. Also in this protocol the feedbacks are transmitted using only one path and not all the available paths for the already mentioned reasons.

This protocol has the advantages of being directly usable in *half-duplex* links because there are never concurrent transmissions and receptions on the same channel and a significant reduction of the overhead in case of optimal choice of the redundancy r.

The number of feedbacks transmitted at each round is very important for the correct functioning of the protocol because the intervals of time where we are allowed to send feedbacks are very limited, and a failed reception of the feedback has negative impacts on the performance of the system; it is possible to modify the default value setting the parameter on the simulator.

In a future research it will be possible for example to improve this protocol by not stopping the transmission on the other paths during the reception of the feedbacks, exploiting the whole potentiality of the links.

3.3.3 Half-duplex protocol with feedback retransmission

The *third protocol* is an extension of the *half-duplex* protocol, that corrects some of its critical issues. One of the problem of the previous protocol is that if after the transmission of a block all the feedback packets are lost then rank of the

decoder matrix stored at the encoder is not updated and in the next round it will retransmit the same number of packets n of the previous round.

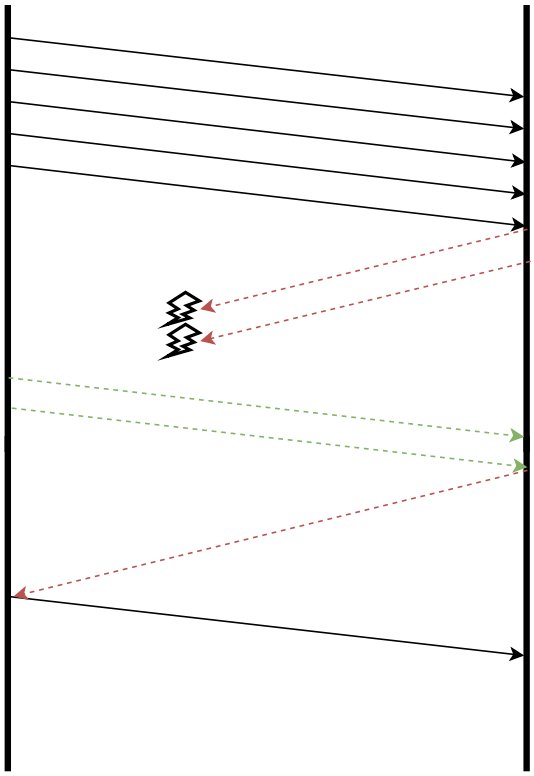


Figure 3.16: Scheme of the third protocol, black solid arrows represent the encoded packets, the red dashed arrows represents the feedback packets, green dashed lines are the *request for feedbacks*.

This protocol is similar to the second but differs for the managing of the case of lost feedback. In more detail:

- If at least a feedback arrives at each round at the source the transmission and retransmission procedures are the same of the the second protocol
- If in a round it happens that all the feedbacks are lost, then the source does not react sending the same number of packets of the previous round but sends a block of *requests for feedback*, that are represented in Fig. 3.16 with the green dashed arrows
- When the decoder receives a *request for feedback* it responds with a new block of feedbacks, returning at the first point

The *request for feedbacks* are very small packets that do not carry information but only a code interpretable by the source, that reacts resending the block of packets relative to the last round.
This protocol aims to minimize the overhead of the transmission induced by the second protocol that in case of loss of all the feedback generates a too big overhead that flood the links of useless packets. As the previous protocol also this can be improved exploiting the empty link during the transmission of the feedbacks in order to send packets of another generation for example, but it is not a trivial modification because it requires a management of packets of different generations and then more memory at the receiver to store packets seen but not decoded.

4

Results

In the previous chapters we have seen the motivations for which we have built a simulator for multipath network coded transmissions, which are the building blocks of this simulator and the algorithms that we have implemented. The simulator consists of a *core* that remains the same in all the versions (explained in section 3.2), and on top of this several protocols (reported in section 3.3) that replicate some of the possible use cases of the `nctun` protocol explained in 3.2. The network that we want to simulate is presented in Fig. 3.5 and is consists of two links, called *forward paths*, that are used simultaneously to transmit network coded information packets, and two *reverse* links that can be used to transmit the *feedback packets*. An important observation that we have previously made is that there are no motivations to send the packets that contain the feedbacks on both the available *reverse links* in a multipath way, incurring in problem of out of order, delayed and lost feedbacks that will degrade the performance of the whole system and complicate the feedback procedure; in addition, the feedback packets are very small as reported in section 3.3 and the throughput boost provided by pooling multiple links is not needed. This considerations leads us to a choice that we have to do in order to employ this *multipath* protocol, which one of the two *reverse* paths we will use to transmit our *feedbacks*. We have seen also in the related works on the NC-MPTCP the importance of a reliable delivery of the feedbacks is important in order to reach good performance, then a good choice of

the link to exploit can drastically change the results reached by the protocol [41]. Our objective is to find a criterion that allows us to choose the best path among the available ones on the base of the information that we have about them, that in our case are the parameters of the link: *Bandwidth, Latency* and *Loss rate*. A first step to do in order to go in this direction is to evaluate the effects of the variation of this parameters on the system performance. We will then simulate the behavior of the developed protocols under different channel conditions, searching the *critical channel parameters* for the feedback transmission.

4.1 Performance evaluation of the full-duplex protocol

We have presented and explained the first protocol in the section 3.3.1 that is thought for *full duplex* links.
The use of Network Coding in our protocols as we have seen brings several advantages in terms of simplification of the feedback management due to the mixing property of this coding scheme and the lack of a *sequential order* of the packets that comes from the same generation. We can also observe how the management of feedbacks in RLNC system is equal for Single Path and Multi Path transmissions, especially in our case where we use a single link to transmit the feedbacks in a multipath setup. The protocols that we have implemented are easily extensible to a single path transmission, by slightly modifying the *scheduler* that will send the packets always on the same path.
Thus, at a first time we have analysed the behavior of the protocols in a single path setup, changing the parameters of the feedback path. In Fig. 4.1 is reported the scheme of the simulated network in this first part, it is composed by a single *forward path* and the related *feedback path*.

In order to evaluate the performance at the modification of the feedback path parameters we have conducted the simulations as follows

- The parameters of the *forward path* remain fixed to the value chosen at the beginning of the simulation from the command line
- Two of the three parameters (bandwidth, latency and loss rate) of the *feedback path* vary among selected intervals
- At each iteration the goodput of the transmission is measured.

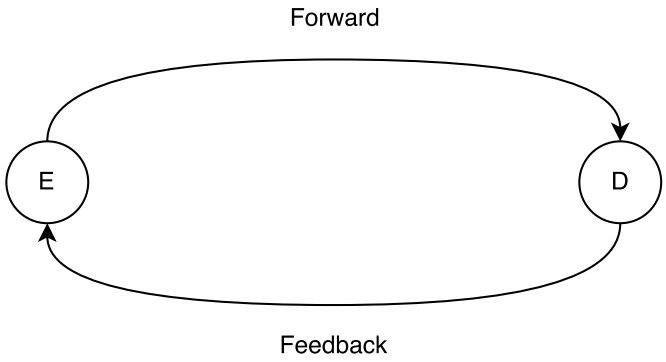


Figure 4.1: Single Path transmission.

The goodput is measured for each generation between the instant of transmission of the first packet t_{tx} and the reception of the feedback that notifies the correct decoding at the receiver t_{feed}, then representing the effective time that has been employed in order to transmit the information of a generation

$$\text{Goodput} = \frac{8 \cdot S \cdot K}{t_{feed} - t_{tx}} \tag{4.1}$$

where S is the `symbol_size` and is reported in Bytes, K is the `number_of_symbols` of the current generation and the 8 is to compute the *Goodput* in bit/s.

If we consider an *iteration* during the simulation as an interval of time in which the parameters of all the links do not change, we can say that during each *iteration* we send several generations, precisely n_{gen}. Thus, the overall goodput of an iteration is computed on the time interval between the transmission of the first packet of the first generation to the reception of the feedback of correct decoding of the last generation. If t_{TOT} represents this amount of time, the overall goodput of an entire iteration will be

$$\text{GOODPUT} = \frac{8 \cdot S \cdot K \cdot n_{gen}}{t_{TOT}} \tag{4.2}$$

Usually each iteration is repeated multiple times (n_{it} iterations) in order to average on the possible realizations of the random processes that are present in the transmission as the random generation of the linear combination contained in the encoded packet and the erasures along the channel propagation; the measured goodput after n_{it} iterations is then the arithmetic mean of the goodput obtained

in each of the iterations computed with (4.2).

The parameters about the network coding scheme are maintained fixed for all the simulations that we have performed, unless not directly specified

- The `number_of_symbols` K of each generation is equal to 128.
- The `symbols_syze` S is equal to 1500 Bytes, similar to an Ethernet connection.

In the following plots we will analyze the obtained goodput of the single path connection of Fig. 4.1 with the *forward path* characterized by a *bandwidth* of 10 Mb/s, *latency* equal to 10 ms and *no losses* on the transmission. Regarding the *feedback path*, we have simulated three different scenarios in which we can compare the effects of the modification of the channel parameters, two by two.

- In Fig. 4.2:
  - Latency: 5 ms fixed
  - Loss rate: 0% to 90%, step 10%
  - Bandwidth: 1 Mb/s to 10 Mb/s, step 1 Mb/s
- In Fig. 4.5:
  - Bandwidth: 10 Mb/s fixed
  - Latency: 5 ms to 50 ms, step 5 ms
  - Loss rate: 0% to 90%, step 10%
- In Fig. 4.4:
  - Loss rate: 0% fixed
  - Latency: 5 ms to 50 ms, step 5 ms
  - Bandwidth: 1 Mb/s to 10 Mb/s, step 1 Mb/s

This means that we have kept fixed a parameter to a default value and changed the other two always in the same intervals.

In Fig. 4.2 we can see a stable plot on the axis of the bandwidth, underlining that at each value of the loss rate the decreasing of the bandwidth of the feedback

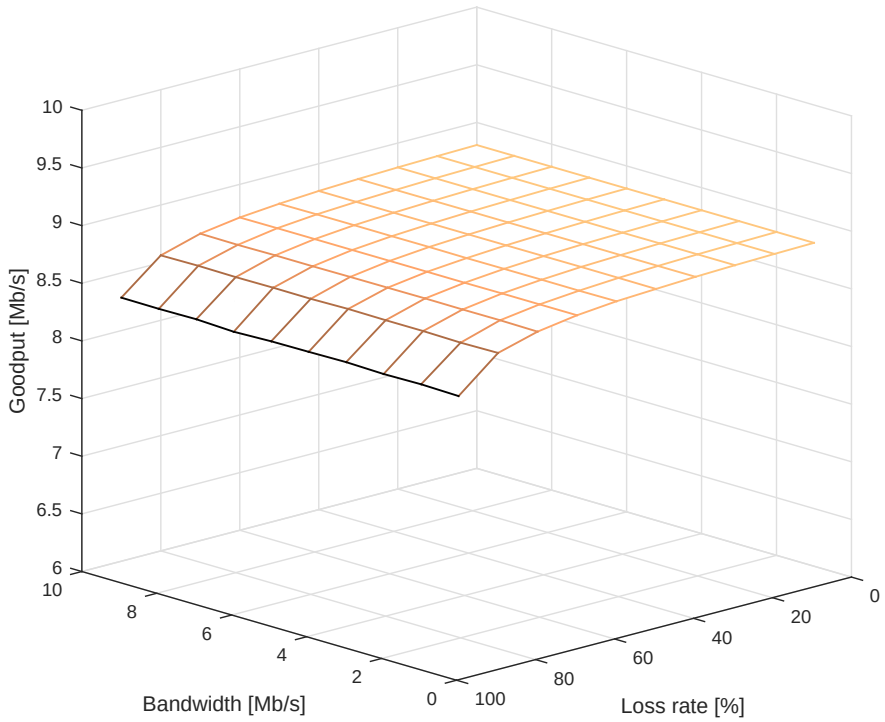


Figure 4.2: Single Path, Bandwidth vs Loss Rate, 10 Mb/s, 0% Loss, 10 ms Latency.

path does not imply a change in the system performance. On the other axis, at the increasing of the of the loss rate, there is a perceivable decreasing of the performance only at very high value, over the 80 %. This plot tells us that the first protocol does not require much bandwidth to transmit in a useful time the feedbacks, meaning that the lightweight design of the feedback packets reached the prefixed goal. In addition the feedback management is robust to channels with very high loss rate, due to the fact that we transmit a feedback for each received packet.

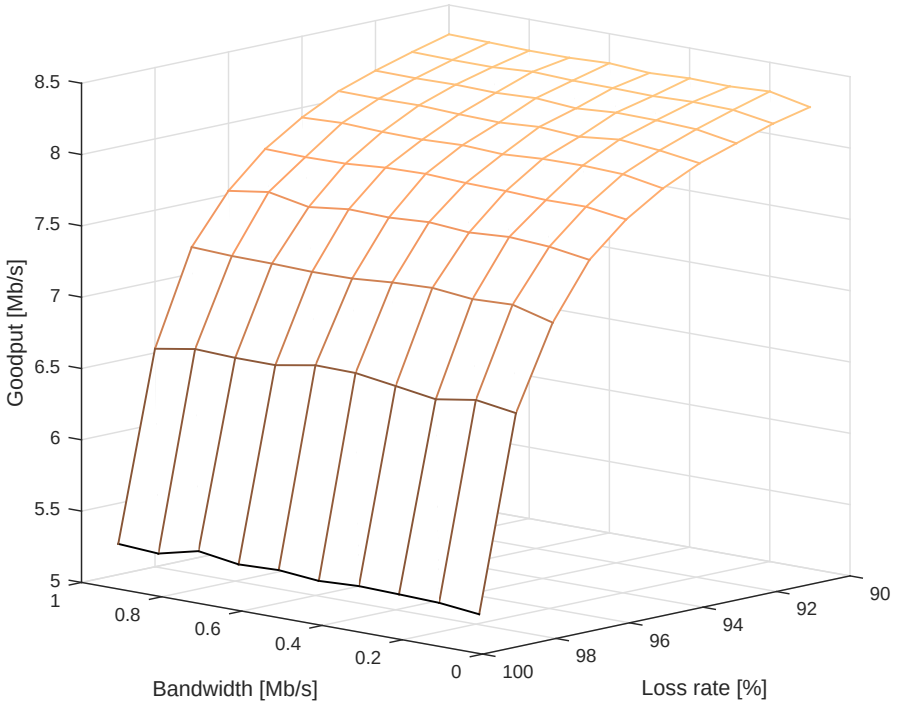


Figure 4.3: Single Path, Bandwidth vs Loss Rate, 10 Mb/s, 0% Loss, 10 ms Latency.

These good results lead us to further stress our system increasing the loss rate and decreasing the bandwidth with respect to the previous case. This experiment is reported in Fig. 4.3 where we set the bandwidth between 0.1 and 1 Mb/s (with steps of 0.1 Mb/s) and the loss rate is between the 90% and 99%. Also in this case, we can see that the performance is stable for all values of the bandwidth that proves that the feedback transmission is not bandwidth constrained. The

same does not happen on the axis relative to the loss rate, here the heavy rate of lost feedbacks degrades the resulted goodput but not as much as expected, in fact we obtain a goodput between 5 and 5.5 Mb/s when the feedback path is set at a loss rate of 99%; it is a good result considering that a channel with such an high loss rate is a very extreme case. The degrading of the performance with the increasing of the loss is principally due to the loss of the last feedback of the generation that acknowledges the decoding of the information and this implies the transmission of further useless packets and the waste of useful time and resources of the *forward path*.

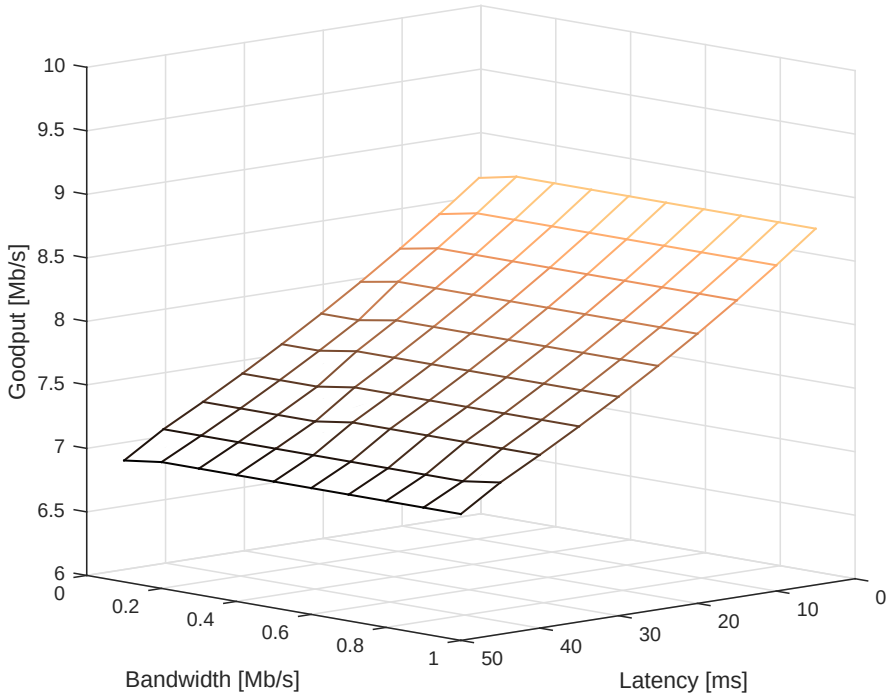


Figure 4.4: Single Path, Bandwidth vs Latency, 10 Mb/s, 0% Loss, 10 ms Latency.

The second evaluation is made by varying of the *bandwidth* and the *latency*, reported in Fig. 4.4. As in the previous case at a fixed value of the latency the goodput is stable for each bandwidth value, proving that also when the latency increases the bandwidth does not become a critical parameter for the feedback transmission. On the contrary, when we increase the latency of the transmission on the feedback path the goodput degrades very rapidly going under the 7 Mb/s

at 50 ms of delay. This is due to the delayed reception of the last feedback at the encoder that produce an *high overhead* and degrades the performance.

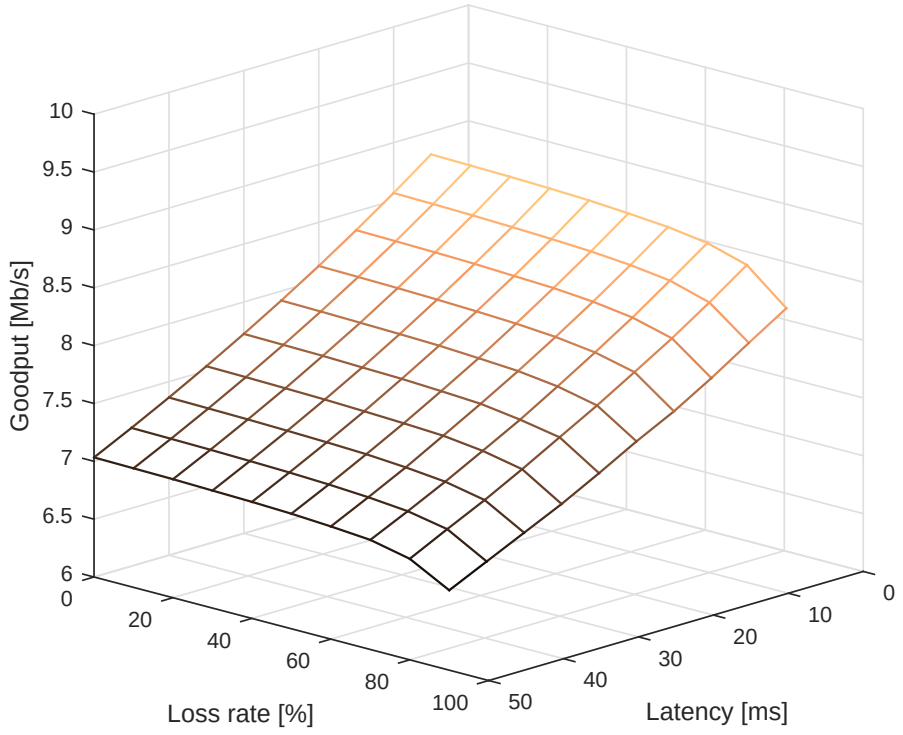


Figure 4.5: Single Path, Loss Rate vs Latency, 10 Mb/s, 0% Loss, 10 ms Latency.

The third and last of this set of comparisons is the one in Fig. 4.5 that represent the third combination of the parameters. Here we have evaluated the performance of the system at the changing if the *loss rate* and the *latency* of the feedback path. The behavior on the axis of the loss rate is as we have seen in the first example, the performance starts to slightly degrade when the rate is higher than 80%; at 90% the goodput is decreased of only 0.5 Mb/s with respect to the case with 0% of loss rate. On the side of the *latency* we can see an evident degradation of the goodput at the increasing of the delay of the feedback path. As in the second case the performance sufferes of a decrease of the goodput of approximately 2 Mb/s going from 5 ms to 50 ms of latency that is not negligible excursion.
With this first set of simulations based on a *single path network coded transmission* we have evaluated the impact that can have a *non perfect feedback path* on the

contrary of what is assumed by the majority of the works related to our. The first results show this is not an acceptable assumption because feedback packets that are transmitted over a non-zero-delay link affect the overall performance of the system. From the last examples we can extract some considerations:

- The *bandwidth* of the feedback path seems to have no incidence on the performance of the system. This a good result, because this means that we have *well designed* the feedback packets and their transmission, and that we have not to care about the bandwidth of the path because the feedback packets will be delivered reliably in a useful time also with low transmission rates
- The *loss rate* is negligible in this protocol apart at very high value, meaning that the protocol is robust enough to feedback losses.
- Since the very critical parameter for the feedback transmission seems to be the *latency*, a small increase results in a rapid degradation of the performance. This result leads us to think that if we can choose between two feedback paths we will choose the one with the lower latency, disregarding the other characteristics.

These first simulations on the single path setup are useful to focus our attention on the impact that the feedback has on the transmissions that use this protocol, whereas the complexity added by the multipath protocol that can be misleading. Now we have a clearer view of how the parameters have a different weight on the transmission of the feedbacks, and we can better understand the results obtained in a multipath case.

4.1.1 MultiPath evaluation

The first difference between single path and multipath transmissions in the presence of the entity called *scheduler*. As explained in section 3.2 the *scheduler* that we have implemented sends a packet on one of the channels every time that channel has finished the previous transmission. In Fig. 4.6 the network utilized for the simulations is reported consisting of two *forward paths* and two *reverse paths*; we remind that only one of the two reverse paths is effectively used to transmit the feedbacks.

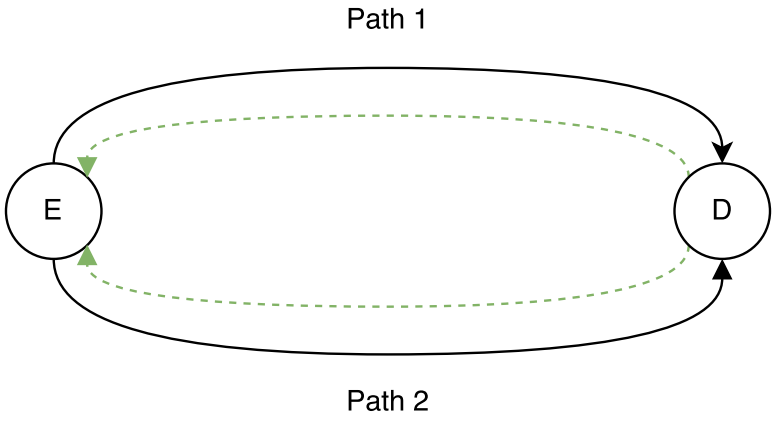


Figure 4.6: Multipath simulated network.

Then we have four paths to simulate in this case, possibly with different parameters; in order to simplify the comprehension of the results we have thought to fix the parameters of the forward path and the related reverse path to the same value.

The plots that we will show are the results of simulations conducted following these steps:

- We select only one parameter that we want to vary. The other parameters are maintained fixed to the values selected from the command line.
- The whole procedure is repeated twice, the first time the selected parameter varies only on the first path, the second time on the second path. The path that not varies, e.g. the second path during the first iteration, remains fixed to the initial value of the other path.
- At each value of the selected parameter is measured the goodput of the transmission, computed as in the single path case with the expressions 4.1 and 4.2.
- In the plots there are a *blue line* that represents the goodput obtained at the first iteration of the simulation, and a *red line* that represents the goodput of the second iteration.

The multipath simulation needs the introduction of two more parameters to set at the beginning

- The most important is the selection *feedback path*, that remains fixed during the whole simulation

- The second is the value that has to assume the selected parameter on the path that has to remain fixed. Without any particular specification, the value is set to the initial value of the other path.

We have then performed some simulations similar to the ones in the single path, in order to see if the considerations that we have made in that case are still valid in a multipath transmission. In this case, remember that the two path consist of a forward and a reverse paths with the same parameters. The *feedback path* is always the first in these evaluation. In Fig. 4.7 we have analysed the protocol at the variation of the bandwidth between 1 Mb/s to 10 Mb/s. The other parameters are 0% of loss rate and 5 ms of latency on both the links. For example, in the first of the two iterations the first path has a bandwidth that changes from 1 to 10 Mb/s while the second is always fixed to 1 Mb/s; all the other parameters in both the links are fixed.

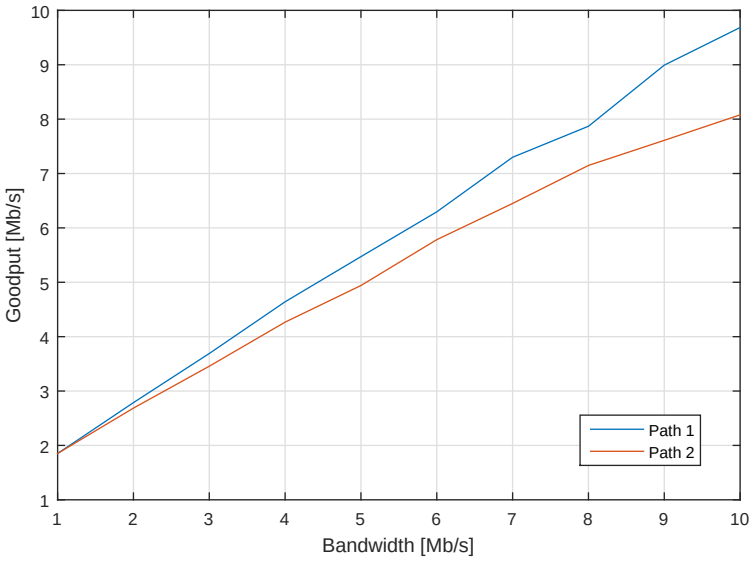


Figure 4.7: Multi Path, Bandwidth, 0% Loss, 5 ms Latency on both the links.

We can see in Fig. 4.7 that the two lines do not overlap although the two path in both the simulation are practically symmetric. The only difference between the

two paths is that the first carries the *feedbacks* on the *reverse path*. We can see that the blue line is always slightly higher than the red line meaning that faster feedbacks gives a little boost in the performances of the system. The difference between the two lines increases at higher value of the bandwidth because in the second iteration the feedbacks arrives too slowly with respect to the increasing transmission speed of the other link. In addition, at high bandwidth values the difference between the two links becomes relevant and the simulation shows the common limitation of the multipath protocols that we have called the *head-of-line blocking* problem, explained in Section 2.4 [3]; in the first case the problem is slightly compensated by the faster feedbacks, but not in the second case, where the feedbacks are transmitted always at the same rate. In conclusion, the goodput is different in the two cases but the difference is not very relevant, in particular if the main difference seems to be due to the head-of-line blocking.
In this simulation as in the following the *statistical confidence intervals* of the results have been evaluated; these intervals in all the situations are very small, then we have decided to do not insert them in the plots. Even in the figures where the *blue* and the *red* lines are very close, the *confidence intervals* with a confidence level of 95% do not overlap.

The second simulation considers a variable loss rate. The values chosen are similar to the ones used in the single path case in order to compare the results; the loss rate varies between 0% and 90%, while bandwidth and latency are fixed on both links to 10 Mb/s and 5 ms, respectively. In Fig. 4.8 the performance decreases when the loss rate of one of the links increases, we can see how the NC scheme maintains the goodput sufficiently high also at high values of the loss rate because NC makes all the received packets useful also if received out of order or with some erasure during the transmission. The most interesting result is the fact that the two lines overlap, meaning that a *noisy feedback path* does not imply a decrement of the goodput obtained with this first protocol. As in the previous case it is possible to see a faster decreasing of performance when the loss rate of the *feedback path* exceeds the 80%, after this value the blue line goes below the red one. This again shows that the feedbacks employed as in the first protocol are sufficiently robust to losses and the loss rate is not a critical parameter.

In the third case we have simulated the goodput versus the *latency*, for the better telling in an interval from 5 to 50 ms; *bandwidth* and *loss rate* are maintained

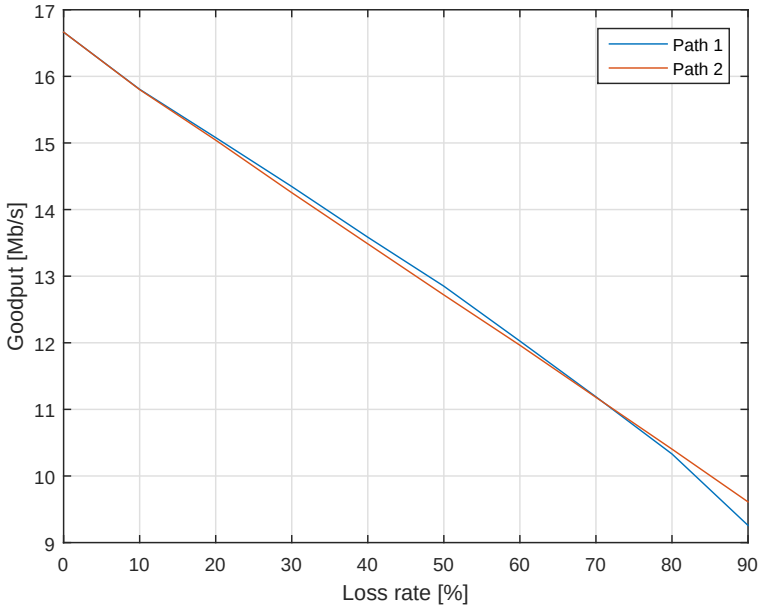


Figure 4.8: Multi Path, 5 ms Latency, 10 Mb/s on both the links.

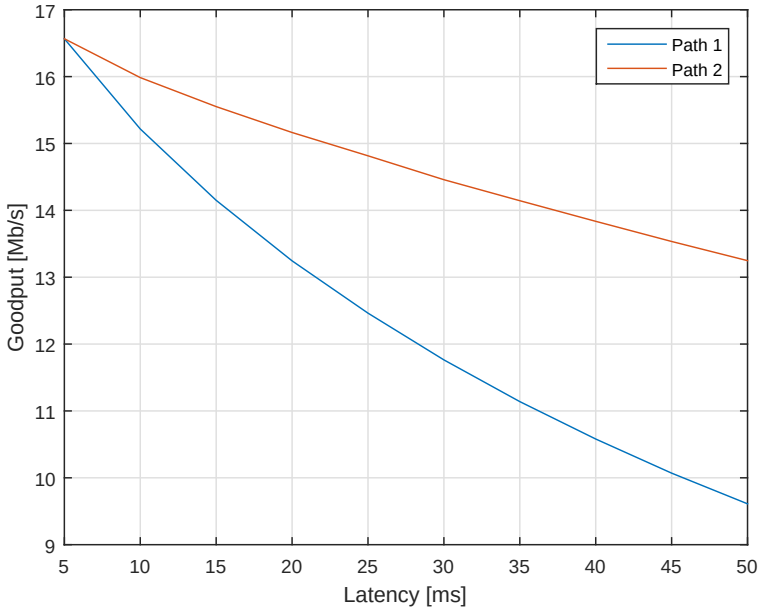


Figure 4.9: Multi Path, Latency, 0% Loss, 10 Mb/s on both the links.

at 10 Mb/s and 0% respectively. In the plot reported in Fig. 4.9 the goodput of both the iterations (i.e. the goodput reported by both the lines in the plots) decreases when the delay increases as we can expect, but the *red line* in Fig. 4.9 is always above the *blue line*. In this case, there is a large difference between the two lines in the plot that underlines what we have found before, that is the latency is the critical parameter for the transmission of the feedbacks. When the *feedback path* is the one that has the lowest delay we obtain a higher goodput with respect to the other case. If we can choose between two paths then we will always select the one with the lowest latency. We will provide some other proofs of this in the following simulation where we have further complicated the configuration of the network.

4.1.2 Extreme cases

The previous simulations suggest that the latency is the parameter, among the three that we have taken into account, that has the largest impact on the performance of the feedbacks and consequently on the whole system. Our goal is now to understand if this conclusion is always valid, e.g. in the case that the path with the *lowest latency* has a *very high loss rate*. In this cases, the choice of the feedback path between the ones available is not trivial, for this reason we provide two extreme situations. These examples are a bit more complex of the previous ones because the two paths are not symmetric and we have to make some more comparison between the results. The first is reported in Figures 4.10 and Figures 4.11 where we have the links with the parameters reported in Table 4.1, and a *latency* that changes as in the previous simulations from 5 to 50 ms on a path at a time.

	Bandwidth	Loss rate
Path 1	**10 Mb/s**	0%
Path 2	**50 Mb/s**	0%

Table 4.1: Parameters of the links in Figures 4.10 and 4.11.

This test aims to clarify if it is more convenient to choose a *feedback path* with a high bandwidth but also a high delay or one with a significantly lower bandwidth and better latency. For these reasons, we have performed twice the simulation

on the same network, switching the designed feedback path from the first to the second link.

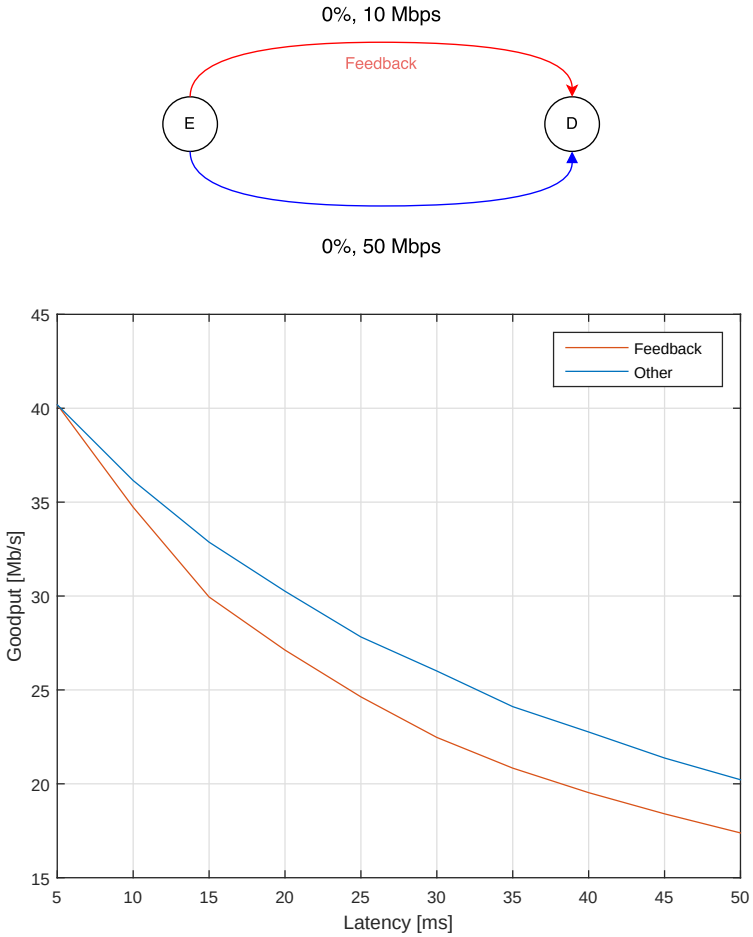


Figure 4.10: Multi Path, Latency, $[10\ 50]$ Mb/s, 0% Loss on both the links. Feedback on the first path.

In the figures,the red plot describes the goodput when changingthe feedback path; the blue line when we change the other path.
The key consideration to make is that in both the plots the higher line is the *blue* one, meaning that disregarding the bandwidth of the path we obtain better results when the latency on the feedback path is the lowest possible. The *blue line* represents the case where the feedback path has a latency fixed to 5 ms while in the other path grows up to 50 ms, and then apart from the initial point the delay of the feedback path is always lower. In Fig. 4.10 the results are better than in Fig. 4.11 because we are using the feedback path with the lowest delay

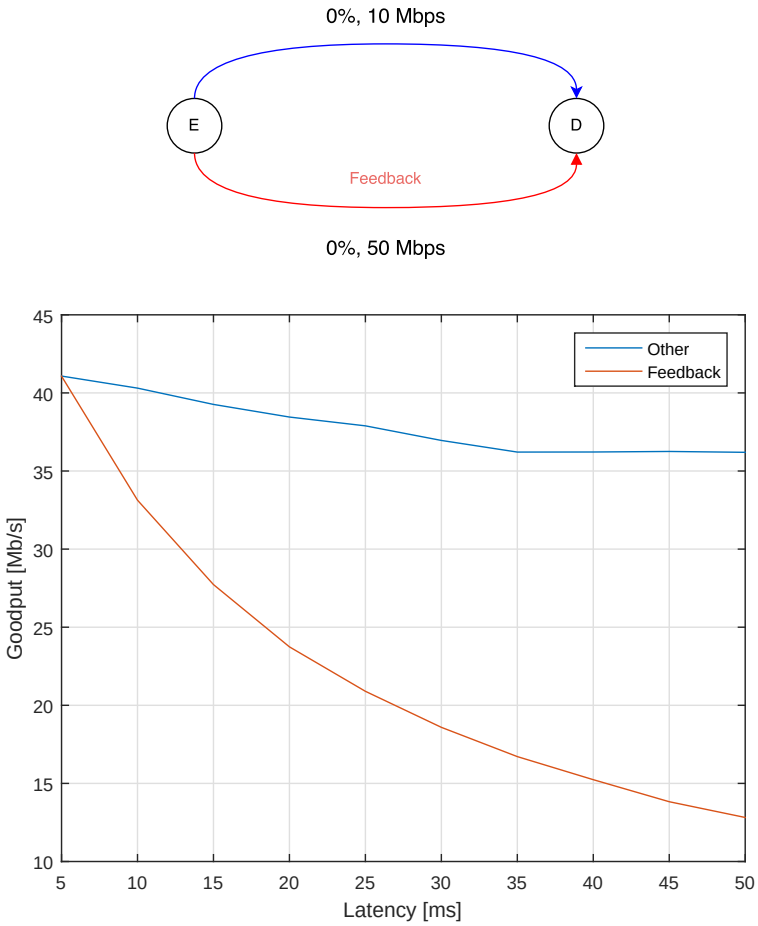


Figure 4.11: Multi Path, Latency, $[10\ 50]$ Mb/s, 0% Loss on both the links. Feedback on the second path.

and the highest bandwidth.
The second comparison that we have tested represents the case where the two paths have a *high loss rate difference* and also a different *latency*. The parameters of the links are reported in the Table 4.2, while the *latency* changes as in the previous case.

	Bandwidth	Loss rate
Path 1	10 Mb/s	**0%**
Path 2	10 Mb/s	**50%**

Table 4.2: Parameters of the links in Figures 4.12 and 4.13.

The procedure performed is the same of the previous case, where we have done two simulation runs switching the feedback path. The results are reported in Figures 4.13 and 4.13 and and we can see that also with this setup the *blue line* is the higher one in both cases. We can conclude that also in the case where a path has a high loss rate is preferable to choose it as feedback path if it has the lowest available latency.

4.1.3 Automatic switching

In the previous sections, starting from the single path cases and step by step increasing the complexity of the simulations we have extracted always similar conclusions from the results, that *latency* is the critical parameter in terms of *feedback path* selection, while *bandwidth* and *loss rate* are of secondary importance. In order to conclude this section of results about the first protocol we want to show how the performance of the system can improve when we select always the best possible path to send the feedbacks.

	Bandwidth	Loss rate
Path 1	10 Mb/s	**0%**
Path 2	10 Mb/s	**80%**

Table 4.3: Parameters of the links in Fig. 4.14.

In Fig. 4.14 we report the results of a simulation where the two links have the parameters reported in Table 4.3; in the first plot, the feedbacks are transmitted

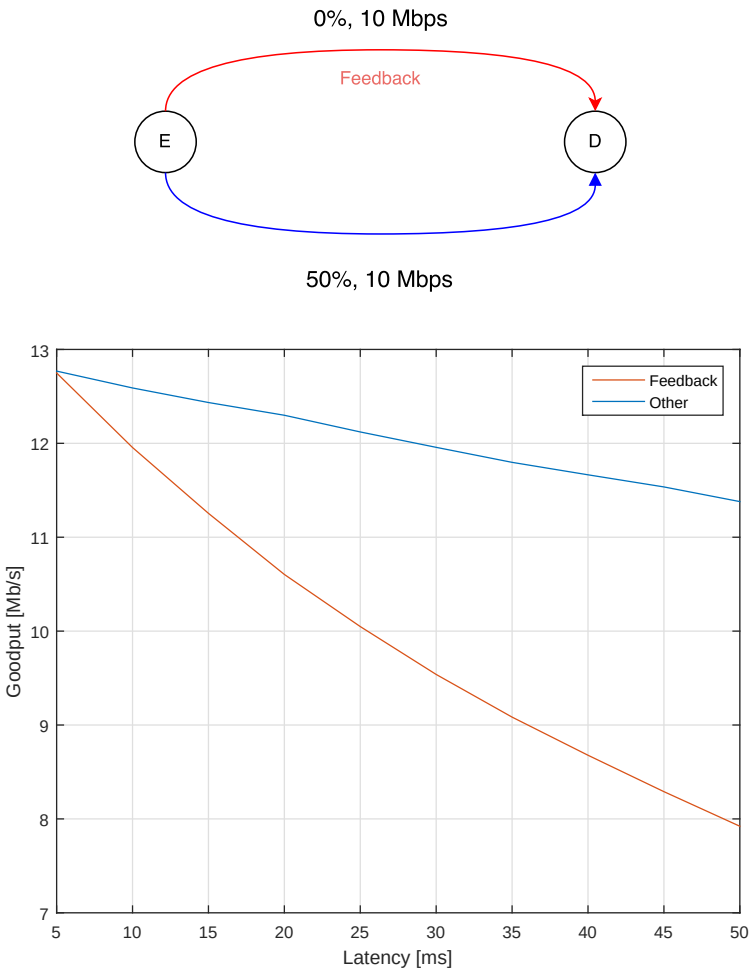


Figure 4.12: Multi Path, Latency, $[0, 50]$% Loss, 10 Mb/s on both the links. Feedback on the first path

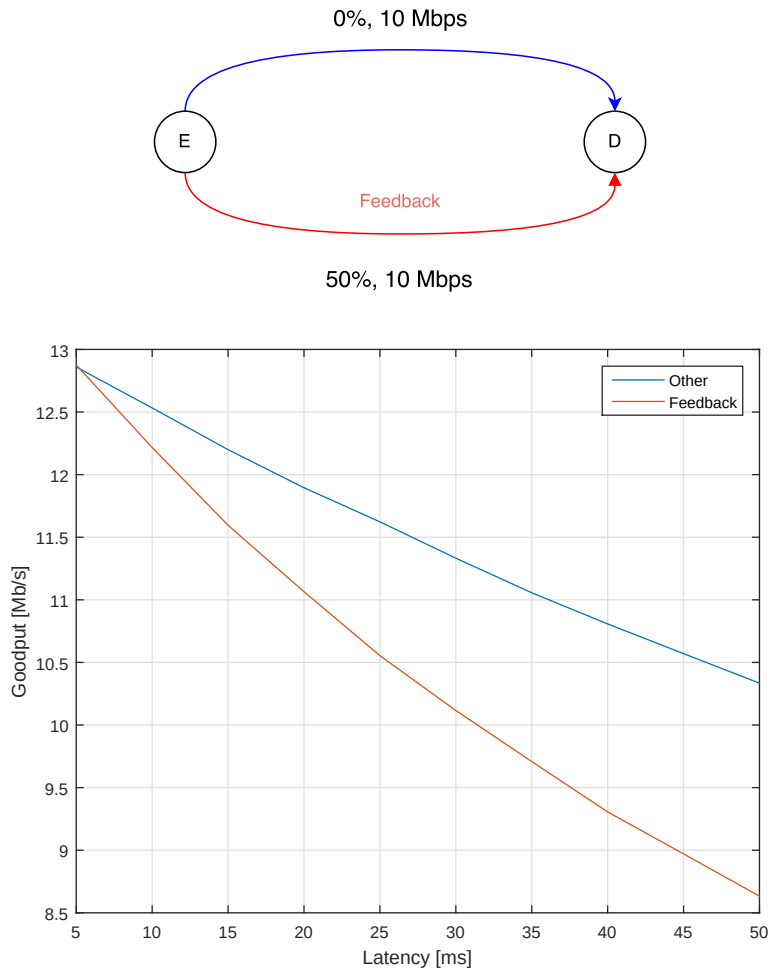


Figure 4.13: Multi Path, Latency, $[0, 50]$% Loss, 10 Mb/s on both the links. Feedback on the second path.

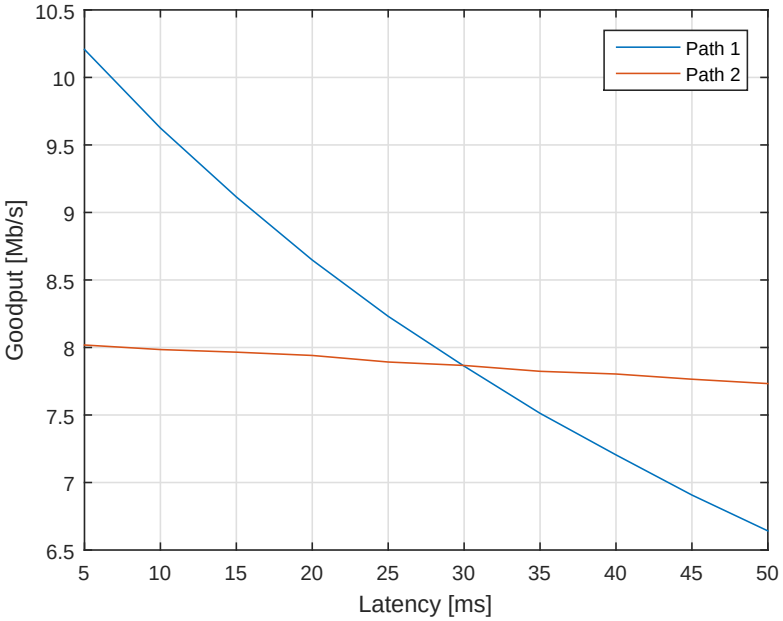
on the first path in the second they are transmitted always in the one with the lowest latency. In these simulations the *latency* changes in the same way of all the other cases that we have analyzed but in the path that remains fixed it is set to 30 ms and not to 5 ms as usual. We can immediately see how in Fig. 4.14(b) the red line is significantly higher before the 30 ms with respect to plot (a), and the same is also for the blue line after the threshold of the 30 ms. In (a) is reported the performance with the better of the two paths in terms of *loss rate*, if we had taken the second path as feedback path the results would have been worse and then the gain in (b) would have been higher. This is an important result because it proves that our considerations were correct and with this technique we are able to exploit the the full potentiality of the links only *setting as feedback path* the one with the lowest delay without taking care of the other parameters.

4.2 Performance evaluation of the half-duplex protocol

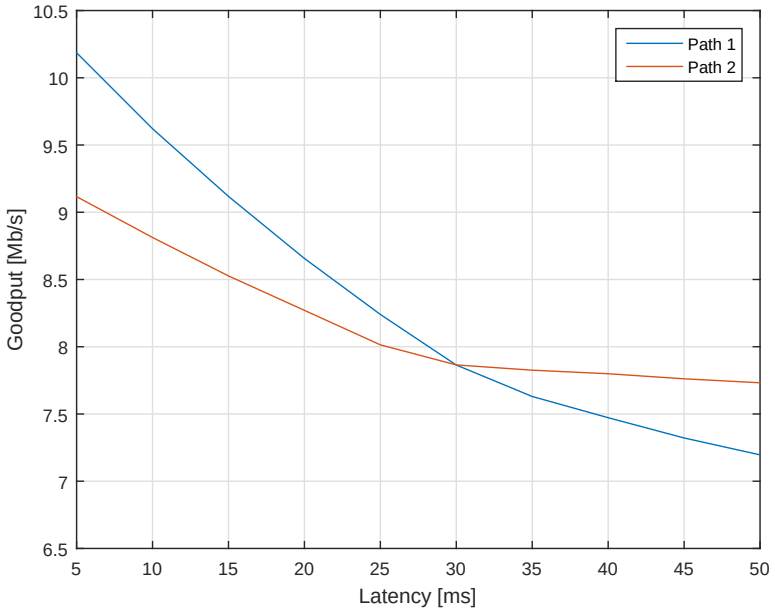
In Section 3.3.2 we have introduced a *second protocol* that differs from the first for it does not send packets and receive feedbacks in the same time but in separated instants as on a *half duplex* medium. In addition it aims to reduce the overhead transmitting only the useful amount of packets and waiting for the feedbacks. In order to optimize the transmission performance tackling the erasures of the links, we want to send a number of encoded packets sufficient for allowing the decoding in the first *round* and also enough feedbacks to notify the state of the decoder at the transmitter.

4.2.1 Redundancy analysis

The second protocol needs to tune the amount of redundancy encoded packets: low redundancy implies multiple rounds, too high redundancy implies overhead, thus in both cases a wrong choice of the redundancy have negative effects on the obtained goodput of the system.
We want to present an example of this scenario in Fig. 4.15. In this plot we have simulated a single path network exactly equivalent to the one utilized in the simulations of the first protocol, with the *forward path* and the *feedback path* that can assume different parameters values. The *forward path* is characterized by a bandwidth of 10 Mb/s, 50% of losses and 10 ms of latency, while the *feedback path*



(a) Feedbacks on the first path.



(b) Automatic switch of the *feedback path*.

Figure 4.14: Automatic switching on the best *feedback path*.

has a latency of 10 ms and loss rate and bandwidth that varies. The literature suggests a redundancy of $\frac{1}{1-\varepsilon}$ that in this case would be equal to 2, but we have tried to set the value to 1.2. On the *feedback path*, the loss rate changes during the simulation, we have then decided to adapt the number of feedbacks per round using (4.3).

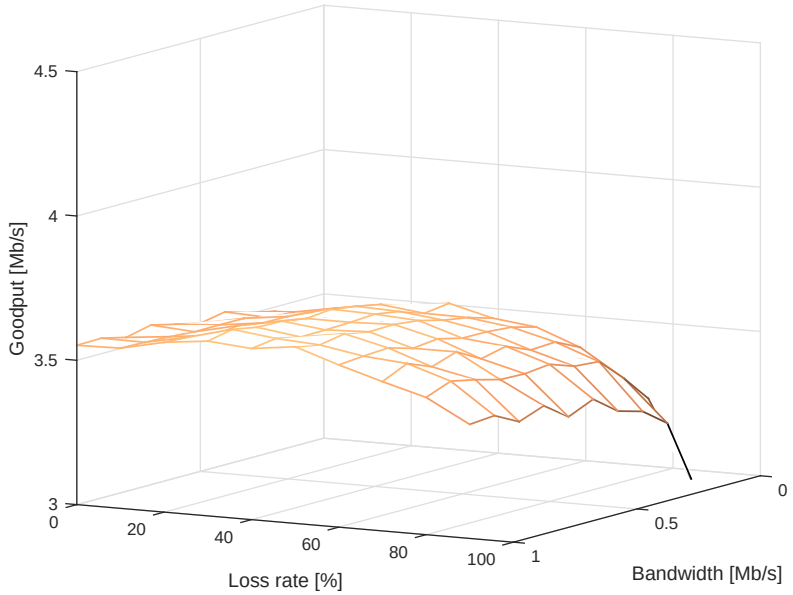


Figure 4.15: Single Path, Low redundancy, 50% Loss, 10 Mb/s, 10 ms.

In Fig. 4.15 we can see that the goodput is stable around the 3.5 Mb/s apart when the loss rate is very high and bandwidth very low; in this case, the performance degrades to 3 Mb/s. The fact that the goodput is stable means that adaptive calculation of the number of feedbacks in function of the loss rate is effective, and that NC maintains the amount of information carried to the decoder stable even when the erasure rate is high as in this case. We can reach higher performance simply setting the right amount of redundancy to send, a low redundancy implies several rounds of transmission. In Fig. 4.16 we have performed the same simulation setting the redundancy to 2.

The resulting goodput is significantly higher than the previous case, especially when the loss rate is under 60%; on the other axis, as expected from the first protocol, the *bandwidth* does not impact on the performance also if it is very

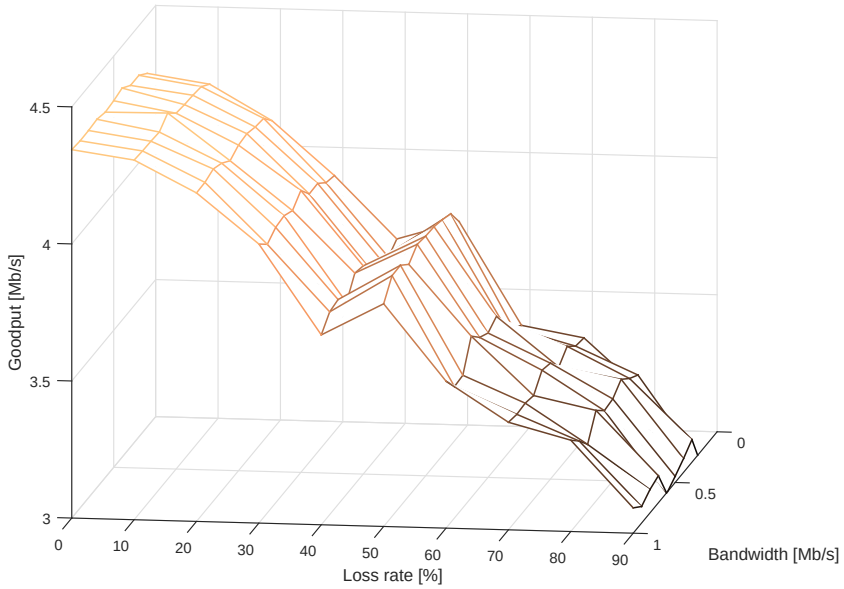


Figure 4.16: Single Path, Right redundancy, 50% Loss, 10 Mb/s, 10 ms.

restricted as in this case that goes from 0.1 to 1 Mb/s. In Fig. 4.16 we can see that there is *jump* in the throughput passing from 40% to 50%, that could be not reasonable in a first analysis. This is due to the fact that we compute the number of feedbacks f as

$$f = \left\lceil \frac{1}{1-\varepsilon} \right\rceil, \quad \text{for } 0 \leq \varepsilon \leq 1 \tag{4.3}$$

and when ε passes from 0.4 to 0.5 the number of feedbacks f passes from 2 to 3 at each round and this implies a little boost of the performance. This consideration tells us that the *optimal value* of f suggested also in [2] is too conservative, for this reason we have made another simulation, reported in Fig. 4.17 , where f is computed adding one to the previous case

$$f = \left\lceil \frac{1}{1-\varepsilon} \right\rceil + 1, \quad \text{for } 0 \leq \varepsilon \leq 1 \tag{4.4}$$

With this small modification to the feedback number we can increase the robustness of the protocol to the losses on the feedback path as we can see in Fig. 4.17 the plot is smoother with respect to 4.16 and the goodput decreases more slowly at the increasing of the loss rate. Looking at this good result obtained

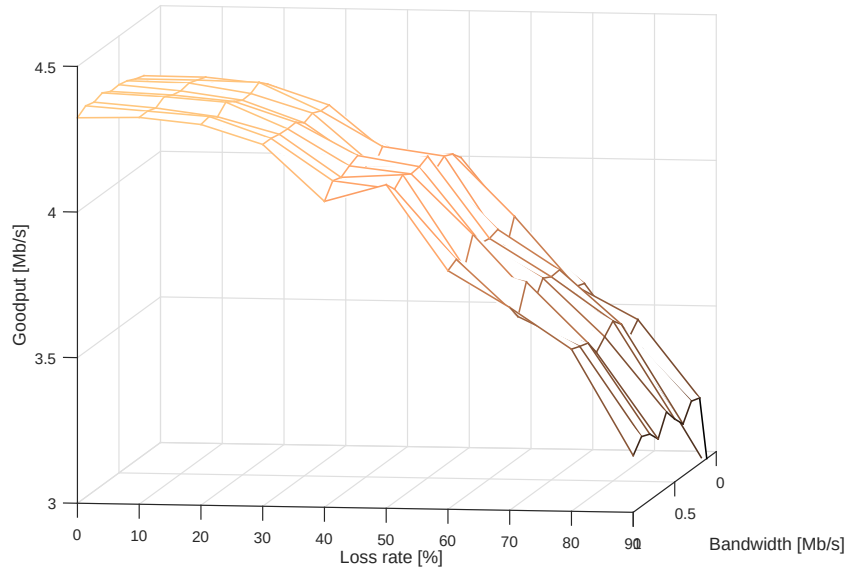
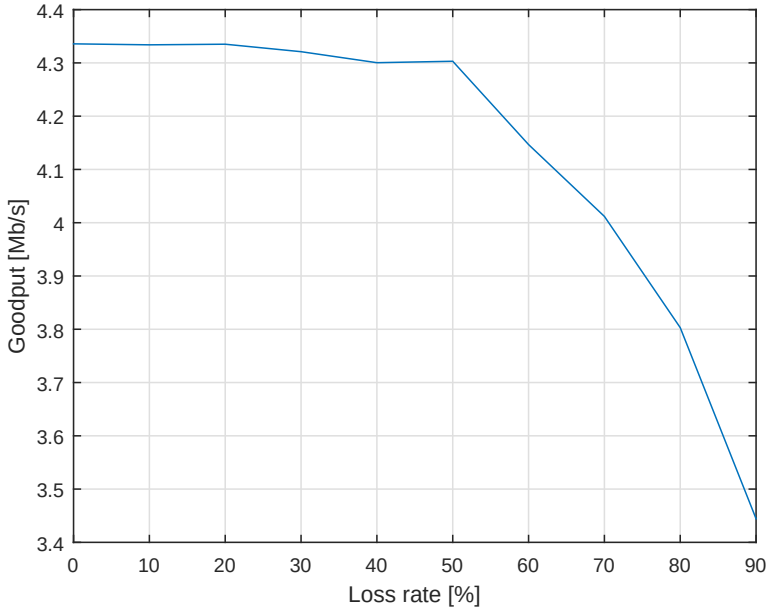


Figure 4.17: Single Path, Right redundancy and one more feedback, 50% Loss, 10 Mb/s, 10 ms.

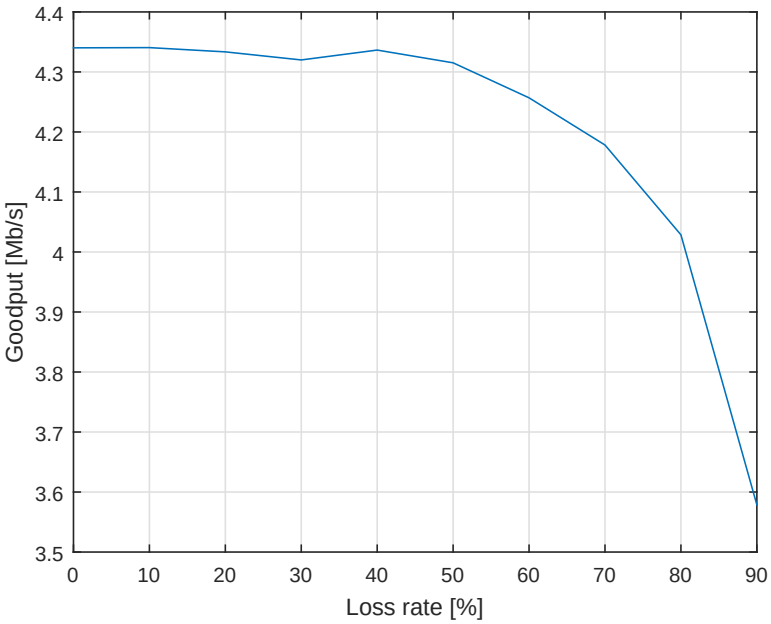
only increasing the number of feedback packets by 1 suggests us to try to add further feedbacks in order to tackle very bad realizations of the erasure process.

Since we have seen that the bandwidth of the feedback path has no relevant impact on the goodput, in Fig. 4.18 we have fixed it at the value of 0.5 Mb/s. In Fig. 4.18 (a) we have sent 3 more feedbacks with respect to the value calculated with 4.3, while in (b) 5 more. In (a) the goodput is maintained over the 4.3 Mb/s until 50% of loss rate and after it decreases rapidly, (b) before 50% has the same goodput of (a) while after the threshold of 50% has a smoother behaviour that maintains the goodput over 4 Mb/s also with 80%. Sending more feedback at high loss rate can maintain a stable goodput for the whole range of loss rates simulated.

In Fig. 4.19 we have tested if the last suggestion was true, doubling the number of feedbacks in the case where the loss rate of the feedback path exceeds 40% that is the point where the plots in Fig. 4.18 begin to decrease the goodput. Doubling the feedbacks for all the loss rate would affect the performances when the rate is low due to useless transmitted feedbacks, while following this procedure we can increase the performances of higher loss rate cases. In Fig. 4.19 we can see how with this strategy the system is able to maintain the goodput stable between 4.4



(a) With 3 more feedbacks per round.



(b) With 5 more feedbacks per round.

Figure 4.18: Single Path, Second protocol, 10 ms, 10 Mb/s, 50% loss.

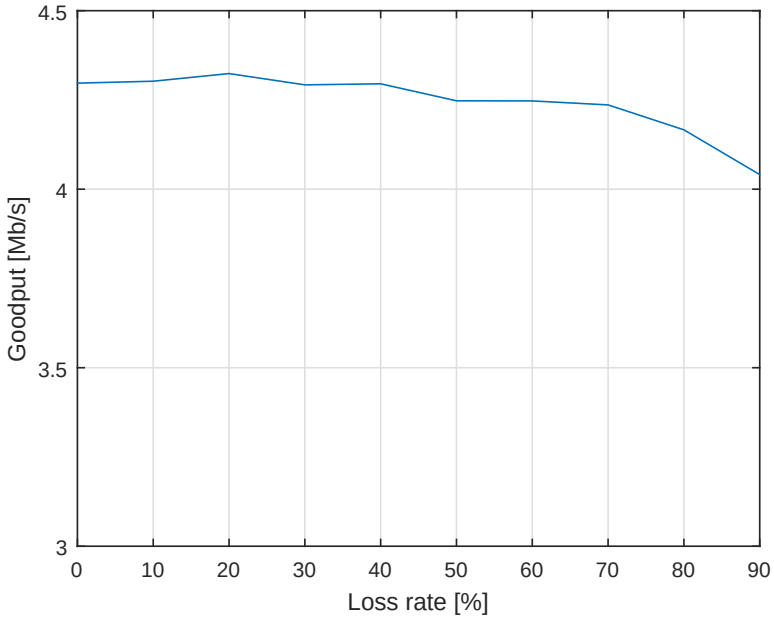


Figure 4.19: Single Path, Right redundancy and one more feedback, 50% Loss, 10 Mb/s, 10 ms.

and 4 Mb/s with loss rate on the feedback path that goes from 0 to 90%.
The last result means that this *half-duplex protocol* can be made *robust to losses on the feedback path* as the first one, carefully choosing a sufficiently high number f of feedbacks. In addition, in general the number of feedbacks computed with (4.3) is not sufficient to have a decent robustness to the feedback erasures and we have to be more conservative.

4.2.2 Multipath evaluation

After this first step where we have analysed the performance of the *half-duplex* protocol in a single path scenario when subject to different amounts of redundancy and number of feedbacks we want to extend the performance evaluation to the *multipath* case. In the previous case we have derived from the results that the bandwidth is not a parameter impacting on the transmission of the feedbacks, while the *loss rate* has a more important role with respect to the first protocol. Here the loss of all the feedbacks of a round degrades significantly the performance but in most case it is possible to work around by simply sending a sufficient number of feedbacks in relation to the erasure characteristics of the link;

it is possible to guarantee a sufficiently high goodput up to the 50% probability of loosing a feedback. With the simulation in the multipath scenario we want to evaluate which is the behaviour of this protocol when the *latency* of the links increases.
The multipath simulation is conducted in the same way of that on the first protocol, where latency of each link increases, on the *forward* and the *related reverse path* together. The parameters employed in the simulations in Figures 4.20 and 4.21 are reported in Table 4.4, while the latency varies from 5 ms to 50 ms with steps of 5 ms.

	Bandwidth	Loss rate
Path 1	10 Mb/s	**0%**
Path 2	10 Mb/s	**50%**

Table 4.4: Parameters of the links in Figures 4.20 and 4.21.

With this simulation, we want to evaluate if in a multipath scenario that employs the *half-duplex* protocol it is still preferable to choose always the path with the lowest delay; as in the last comparisons of the first protocol we have performed the simulation testing both the links as *feedback path*. Since we have two paths with the same bandwidth and one has a loss rate of 50%, we can consider as we have a link with the 25% of loss rate; following the *optimal expression* we should set the transmission redundancy to $\frac{1}{1-0.25} = 1.33$ but in order to be a bit more conservative we have set the redundancy to 1.4. Trying to set the redundancy to 1.6 results in worse performance due to the *overhead*. From the Figures 4.20 and 4.21 we can deduce that in both cases we can get a higher throughput choosing as *feedback path* the link with the *lower latency*, also in the case that this link is the one with the loss rate equal to 50%.
We can finally conclude that in a multipath scenario where we have to choose which of the available *reverse links* use to transmit the feedbacks, we will base our decision only on the latency of those links, selecting the one with the lowest value.

In particular, we want to underline how the high flexibility and adaptability against erasures of these protocols is completely due to Network Coding that allows the system to react to a packet loss with the creation of new encoded

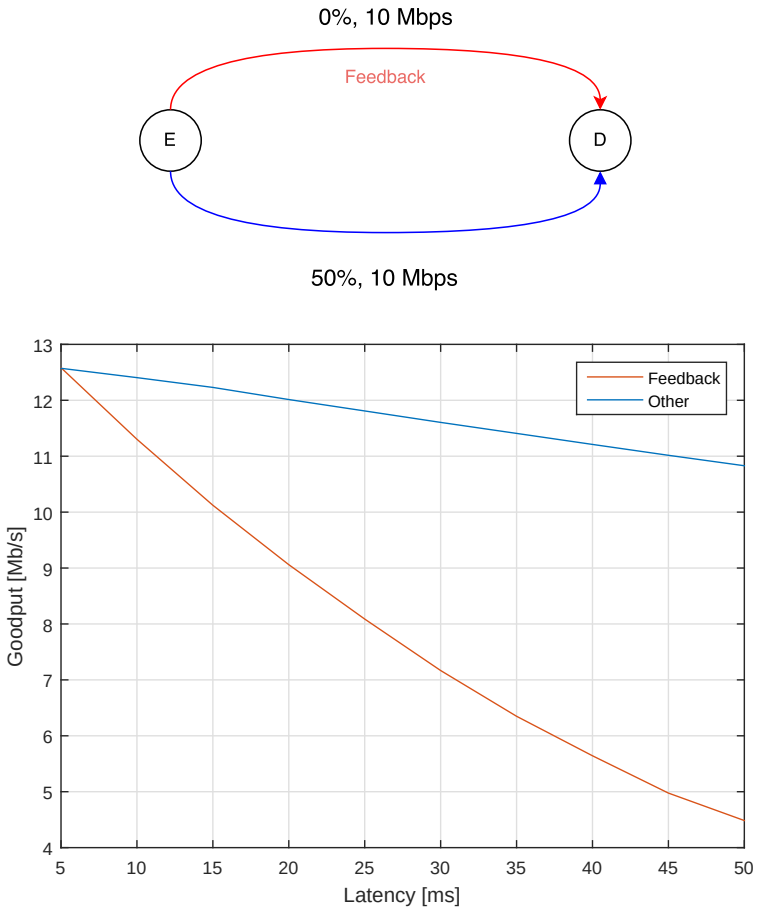


Figure 4.20: Multi Path, second protocol, Latency, $[0, 50]$% Loss, 10 Mb/s on both the links, 1.4 redundancy. Feedback on the first path.

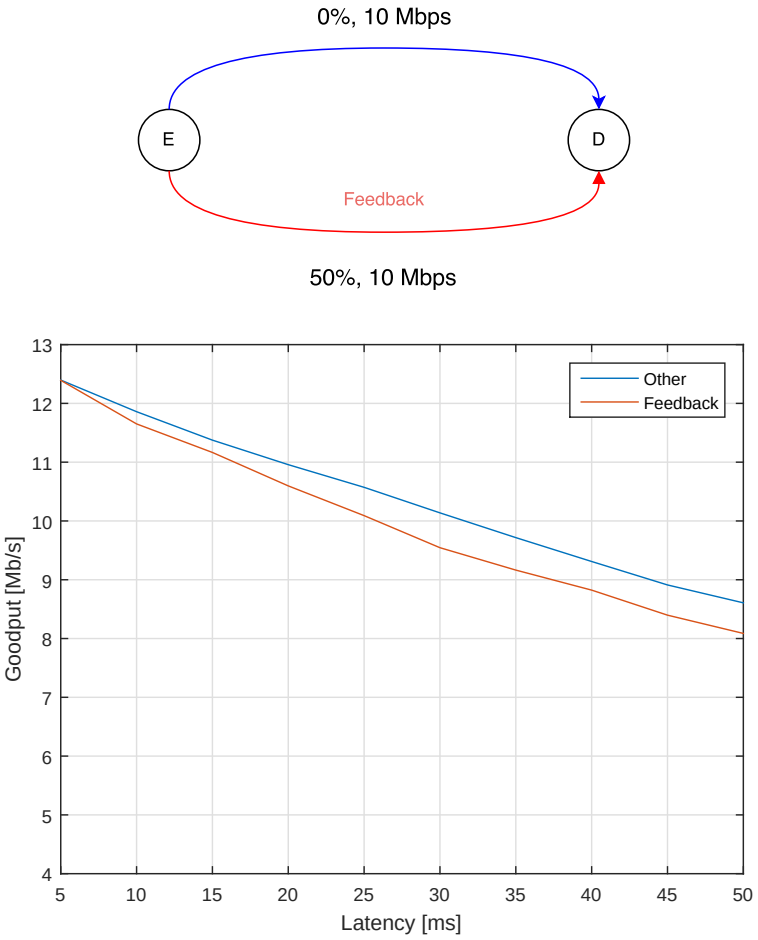


Figure 4.21: Multi Path, second protocol, Latency, $[0, 50]$% Loss, 10 Mb/s on both the links, 1.4 redundancy. Feedback on the second path.

packets disregarding which packet was lost. If we have the information about the loss rate of the links, Network Coding allows not only to *react* but also to *prevent* the erasures of the links, as in the *half-duplex protocol* where we transmit redundant packets in order to avoid the requests for retransmissions that have an high cost in terms of performance. The concept of *redundant packet* has a meaning only in a NC environment due to the fact that is a rate-less code and allows to create an arbitrary number of encoded packets from the same generation that are useful to recover any of the lost packets during the transmission. Network Coding brings important implementation and performance benefits when embedded in a multipath protocol: NC allows to do not keep track of the specific packets that have been transmitted on a link, NC encoder only counts of the number of transmitted packets. Multipath transmission is one of the natural implementations of NC, this coding scheme has been designed to easily split the traffic among several links in order to maximize end-to-end throughput and resilience against losses. We have exploited these features to develop the protocols, and with the simulator we have proved that the aforementioned benefits of NC can be reached. Finally, NC not only allows to simplify the transmission of packets, but also the feedback management; from the simulations, we have deduced that thanks to NC features and the right settings of the parameters, we can make the feedback procedure robust to losses and low bandwidth, increasing the reliability and the performance of the developed protocols, both in single path and multipath scenarios.

5

Conclusions And Future Work

This thesis introduced a simulator for the multipath transmissions that employs the Network Coding scheme to encode packets, and analyzed the performance of two proposed protocols, in particular evaluating the impact that the feedback packets have on the performance.

We first introduced the concept at the base of Network Coding, and then described in detail the main application of this paradigm, called Random Linear Network Coding. The main characteristics of Random Linear Network Coding were then analysed, along with some examples that explain the benefits of this coding scheme as the *capacity achieving throughput* in a multicast scenario, the *rateless encoding* and the *recoding at intermediate nodes*, and the increased transmission robusteness with respect to the classical *store and forward* paradigm [13] [12].

Furthermore, some related works were reported about the *integration* of a *feedback structure* in a network coded transmission protocol [14] [15]; this is not trivial due to the lack of the concept of *sequential ordering* of packets, and a method was proposed that utilizes *degrees of freedom* instead of the traditional *sequence number*.

After the feedback management, the performance and robustness benefits have been discussed that are expected by a *multipath transmission*, implemented on a multihomed device. Besides, we have analysed also the issues of a multipath

protocol and the *schedulers* provided by the literature that aim to solve these problems [33] [32], implemented in MPTCP [8]. Thus, were reported some examples about multipath solutions that exploit the unique features of Random Linear Network Coding to solve the critical issues discussed before [4] [5]. Then, the different solutions and tools were briefly discussed that usually are adopted to develop a protocol as the *emulators* and the *simulators*.

An overview of the Kodo library [1] was provided in order to describe how Random Linear Network Coding is implemented in the simulator and what this tool can offer to the developers that want to exploit the features of NC in their projects. Details on the specific implementation of the multipath simulator were discussed, in particular the *encoding procedure*, the *scheduler* and the simulated *propagation* of the packets along the links. After that, we went in the details of the *reception* and the *decoding* procedures that control the transmission of the feedbacks packets.

The developed protocols were presented next, that integrate the *transmission multipath protocol* and a *feedback management system* based on the NC scheme, and propose different solutions for *half duplex* and *full duplex* links. The use of the Network Coding scheme allowed us to manage the feedbacks and the retransmissions in a flexible and efficient way. As a futher contribution, we showed the results about the feedback path choice.

Before the results, were discussed the reasons that led us to use only one path for the feedback transmission, and from this consideration we derived the need to choose the best of the available paths. In the last section, we researched which characteristics of the paths make a link preferable among the available ones, plotting the performance of the protocols with different links' configurations, in order to deduce how the modification of a parameter (as *bandwidth, latency* and *loss rate*) impacted on the *feedback transmission and effectiveness* and on the overall system performance. The results showed that in both protocols, the *feedback management* is robust and effective also in connections with *low bandwidth* and *high erasure rate*, but that is more affected by high latency links that imply a degrading of the performances. The practical conclusion that we deducted from these results is that the choice of the path on which to send the feedback packets, called *feedback path*, would be based only on the selection of the *lowest latency* disregarding the *bandwidth* and the *loss rate*.

A related consideration has been derived from the results: in general we cannot assume that the feedbacks are transmitted on a *perfect delay-free channels*, this is only a theoretical abstraction that never reflects reality. The majority of works about feedbacks over network coding transmissions uses this assumption to show the theoretical performance of their systems. We saw that the delay applied to the feedbacks has an important impact on the performance of the system and then we cannot consider feasible and realistic such an assumption.

5.1 Future Work

As future work, it would be interesting to include in the simulator more refined schedulers, that exploit the information about the links in order to optimize throughput and reduce the *head-of-line blocking* problem that is present, in a reduced form, also when Network Coding is employed. We think that the develpement of a *multipath scheduler* specific to work together with Network Coding exploiting all its unique features could further reduce the magnitude of this problem that affects the multipath protocols.

Moreover, the simulator can be extended to reproduce intermediate nodes between the source and the receiver exploiting the possibility of *recoding without decoding* specific of Random Linear Network Coding; this solution will probably show further improved throughput and resilience against losses with respect to the end-to-end code that we have employed. In fact, the *feedback protocol* could exploit the recoding operation to require the retransmission of a new encoded packet to the last node and not to the the source.

In addition, the simulation framework developed in this thesis can be utilized to test other NC protocols that can be easily attached to the existing structure, evaluating the overall throughput of the system and not only the impact of the feedbacks.

References

[1] Steinwurf, "Introduction to kodo," 2016. [Online]. Available: http://steinwurf.com/_products/kodo.html

[2] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: theory and implementation," in *Proceedings INFOCOM 2009*. IEEE, 2009, pp. 280–288.

[3] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *Proceedings of ACM SIGCOMM workshop on Capacity sharing workshop*. ACM, 2014, pp. 27–32.

[4] M. Li, A. Lukyanenko, and Y. Cui, "Network coding based multipath TCP," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, 2012, pp. 25–30.

[5] J. Cloud, F. du Pin Calmon, W. Zeng, G. Pau, L. M. Zeger, and M. Medard, "Multi-path TCP with network coding for mobile devices in heterogeneous networks," in *Vehicular Technology Conference (VTC Fall)*. IEEE, 2013, pp. 1–5.

[6] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "Wifi, lte, or both: Measuring multi-homed wireless internet performance," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 181–194.

[7] J. Postel, "Transmission Control Protocol," Internet Engineering Task Force, Request for Comments (Standard) 793, September 1981. [Online]. Available: ftp://ftp.isi.edu/in-notes/rfc793.txt

[8] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses." Internet Requests

for Comments, RFC 6824, January 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824

[9] J. Iyengar, C. Raiciu, S. Barre, M. J. Handley, and A. Ford, "Architectural Guidelines for Multipath TCP Development," RFC 6182, Mar. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6182.txt

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

[11] T. Ho, M. Medard, J. Shi, M. Effros, and D. R. Karger, "On randomized network coding," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 1. Citeseer, 2003, pp. 11–20.

[12] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," 2003.

[13] T. Ho and D. Lun, *Network coding: an introduction*. Cambridge University Press, 2008.

[14] C. Fragouli, D. Lun, M. Médard, and P. Pakzad, "On feedback for network coding," in *Annual Conference on Information Sciences and Systems*. IEEE, 2007, pp. 248–252.

[15] J. K. Sundararajan, D. Shah, and M. Médard, "Feedback-based online network coding," *arXiv preprint arXiv:0904.1730*, 2009.

[16] I. Standardization, "Iso/iec 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model," *International Standard ISOIEC*, vol. 74981, p. 59, 1996.

[17] N. Benvenuto, R. Corvaja, T. Erseghe, and N. Laurenti, *Communication systems: fundamentals and design methods*. Wiley, 2007.

[18] J. Irvine and D. Harle, *Data communications and networks: An engineering approach*. John Wiley & Sons, 2002.

[19] L. Badia, M. Rossi, and M. Zorzi, "Sr arq packet delay statistics on markov channels in the presence of variable arrival rate," *IEEE Transactions on Wireless Communications*, vol. 5, no. 7, pp. 1639–1644, 2006.

[20] L. Badia, "On the effect of feedback errors in markov models for sr arq packet delays," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, 2009, pp. 1–6.

[21] C. Wang, T. Gou, and S. A. Jafar, "Multiple unicast capacity of 2-source 2-sink networks," *CoRR*, vol. abs/1104.0954, 2011.

[22] M. Hundebøll and J. Ledet-Pedersen, "Inter-flow network coding," 2011.

[23] E. Bareiss, *MULTISTEP INTEGER-PRESERVING GAUSSIAN ELIMINATION.*, Jan 1966. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/4474185

[24] J. E. Gentle, *Numerical linear algebra for applications in statistics*. Springer Science & Business Media, 2012.

[25] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, "Network coding for mobile devices-systematic binary random rateless codes," in *IEEE International Conference on Communications Workshops*, 2009, pp. 1–6.

[26] R. Koetter and M. Médard, "Beyond routing: An algebraic approach to network coding," in *INFOCOM. Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, vol. 1, 2002, pp. 122–130.

[27] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," 2003.

[28] D. S. Lun, M. Médard, R. Koetter, and M. Effros, "On coding for reliable communication over packet networks," *Physical Communication*, vol. 1, no. 1, pp. 3–20, 2008.

[29] ——, "Further results on coding for reliable communication over packet networks," in *Proceedings International Symposium on Information Theory (ISIT)*. IEEE, 2005, pp. 1848–1852.

[30] D. S. Lun, P. Pakzad, C. Fragouli, M. Médard, and R. Koetter, "An analysis of finite-memory random linear coding on packet streams," in *International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*. IEEE, 2006, pp. 1–6.

[31] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring mobile/WiFi handover with Multipath TCP," in *Proceedings of ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*. New York, NY, USA: ACM, 2012, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2342468.2342476

[32] O. Bonaventure, M. Handley, and C. Raiciu, "An Overview of Multipath TCP," vol. 37, no. 5, Oct. 2012. [Online]. Available: https://www.usenix.org/publications/login/october-2012-volume-37-number-5/overview-multipath-tcp

[33] S. Barré, C. Paasch, and O. Bonaventure, "Multipath TCP: from theory to practice," in *NETWORKING 2011*. Springer, 2011, pp. 444–457.

[34] ns 3, "What is ns-3?" 2016. [Online]. Available: https://www.nsnam.org/overview/what-is-ns-3/

[35] mininet, "Mininet overview," 2016. [Online]. Available: http://mininet.org/overview/

[36] N. Matloff, "Introduction to discrete-event simulation and the simpy language," *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, vol. 2, p. 2009, 2008.

[37] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*. San Diego, CA, USA: Academic Press Professional, Inc., 1984.

[38] Steinwurf, "Fifi overview," 2016. [Online]. Available: http://steinwurf.com/_products/fifi.html

[39] P. S. Foundation, "Python overview," 2016. [Online]. Available: https://www.python.org/about/

[40] Steinwurf, "Kodo-python," 2016. [Online]. Available: https://pypi.python.org/pypi/kodo/8.0.1

[41] J. K. Sundararajan, "On the role of feedback in network coding," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[42] M. Hundebøll, J. Ledet-Pedersen, J. Heide, M. V. Pedersen, S. A. Rein, and F. H. Fitzek, "Catwoman: Implementation and performance evaluation of IEEE 802.11 based multi-hop networks using network coding," in *Vehicular Technology Conference*. IEEE, 2012, pp. 1–5.

[43] M. Hundebøll, S. A. Rein, and F. H. Fitzek, "Impact of network coding on delay and throughput in practical wireless chain topologies," in *IEEE Consumer Communications and Networking Conference (CCNC)*, 2013, pp. 381–386.

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

[45] E. Drinea, C. Fragouli, and L. Keller, "Delay with network coding and feedback," in *IEEE International Symposium on Information Theory*, 2009, pp. 844–848.

[46] G. Cherubini and N. Benvenuto, "Algorithms for communications systems and their applications," 2003.