POLITECNICO DI MILANO

Corso di Laurea Magistrale in Ingegneria delle Telecomunicazioni
Dipartimento di Elettronica e Informazione

Network Coding for Dense Cooperative Wireless Cloud Networks

Relatore:

Chiar.mo Prof. Umberto SPAGNOLINI

Correlatore:

Chiar.ma Prof.ssa Monica NICOLI

Tesi di Laurea di:

Marco VISIN

Matricola: 783444

Anno Accademico 2012-2013

Abstract

(Italian language)

Questo lavoro di tesi, dai contenuti sia teorici sia sperimentali, propone una soluzione al crescente problema della congestione delle reti mobili. Infatti al giorno d'oggi la vasta diffusione di dispositivi personali, come smartphone, tablet e notebook, dotati tutti di connettività wireless, ha portato ad un massiccio utilizzo delle attuali risorse offerte dalle reti radiomobili. La ricerca si sta concentrando nella direzione di un miglioramento della capacità delle reti radiomobili introducendo nuove tecnologie, come LTE, che offrono prestazioni migliori agli utenti, con lo svantaggio per gli operatori di dover adeguare gli attuali apparati radio disposti sul territorio. Questo lavoro di tesi si inquadra in un ambito di ricerca differente, focalizzato sull'ottimizzazione dei sistemi già esistenti.

L'idea è di utilizzare la teoria del *Network Coding*. In letteratura sono presenti numerosi contributi a riguardo, che sono stati analizzati e riassunti in una prima sezione del lavoro di tesi. Il contributo innovativo è rappresentato dalla definizione di un algoritmo basato sulla teoria del Network Coding, che presenta alcune variazioni che lo rendono particolarmente adatto per essere implementato su dispositivi reali, anche di piccole dimensioni, dotati di bassa potenza elaborativa e di stringenti limitazioni sull'energia a disposizione.

L'algoritmo proposto viene comparato con due algoritmi di riferimento in un ambiente simulato numericamente. In questo modo si ottiene un limite superiore alle prestazioni che possono essere offerte da una eventuale imple-

mentazione pratica dell'algoritmo, nonché un paragone con le performance ottenibili con gli algoritmi attualmente a disposizione.

Una volta terminata la fase di validazione numerica, viene presentata una applicazione basata su piattaforma Android che consente di valutare in un ambiente reale il comportamento dell'algoritmo stesso. L'applicazione consente ad un gruppo di utenti posti in prossimità di cooperare per lo scaricamento di un contenuto remoto comune, quale può essere del materiale informativo da utilizzare ad una conferenza. L'applicazione utilizza tecnologie innovative, quali Wi-Fi Direct, per creare una piattaforma di condivisione sul quale vengono scambiati i contenuti richiesti, utilizzando appositi protocolli definiti in questa tesi che implementano l'algoritmo di Network Coding proposto.

Dall'analisi dei risultati è possibile vedere come l'algoritmo porti sensibili vantaggi e sia realmente applicabile nell'utilizzo quotidiano.

Contents

List of Figures

List of Tables

Glossary

2-by-2 XOR 2-by-2 XOR algorithm. 2, 3, 34, 37, 41–43, 46, 48, 74–76, 80–82, 94

AP Access Point. 11, 52, 53
API Application Programming Interface. 51, 76

CRC Cyclic redundancy check. 57, 62, 69, 74, 77, 91
CSP Cooperation Server Protocol. 57

GLC Global Linear Combination algorithm. 16, 35, 37, 41–43, 46, 48, 49, 62, 81

IP Internet Protocol (IP). It is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables inter-networking, and essentially establishes the Internet. 9, 60, 69, 90

LI Local Intra-Cloud Interface. 6, 7, 15, 24, 27, 28, 30, 33, 36, 41, 43, 45, 47, 52, 73, 74, 76
LITT Local Interface Transmission Time. 28, 30

MAC Media Access Control. In the seven-layer OSI model of computer networking, it is a sublayer of the data link layer, which itself is layer 2 protocol.. 9, 14, 61, 63
MTU Maximum Transmission Unit. 27

NC Network Coding. 1–12, 14–18, 23, 24, 28, 33, 35, 36, 41–43, 45–48, 50, 55, 59, 73, 75, 76, 79–82, 86, 91, 93

NCP Network Coding Protocol. 57, 61, 62
NIC Network Interface Controller. 76, 77, 79

OS Operating System. 51, 52, 60, 81

P2P GO Peer to Peer Group Owner. 54, 69

RI Remote Interface. 6, 7, 15, 24, 27, 28, 30, 32, 36, 39, 43, 45, 48, 57, 71, 73–77, 92
RITT Remote Interface Transmission Time. 28, 30, 32
RLB Random Local Broadcast algorithm. 33, 34, 41–43, 45–48, 74, 75, 81, 94

TCP Transmission Control Protocol. 57, 62, 63, 77

UDP User Datagram Protocol. 60–63, 77

Introduction

The widespread diffusion of low cost smart radio devices (i.e., smart-phone, laptop, tablet), has fostered the diffusion of mobile networks in every location. Mobile networks are nowadays an essential component of the modern life, necessary for every commercial purpose. They carry data to and from personal devices, like smartphones and laptops, but also carry data from sensors, remote control and many other applications. In the future the machine-to-machine traffic will grow exponentially, leading to the collapse of the existing network infrastructure. Wi-Fi networks, indeed, suffer from a severe throughput limitation and do not scale to dense large networks.

The research is mainly focused on mobile network improvements in terms of throughput offered to the users, of minor delay and obviously in terms of costs for the equipments purchase and deployment. This is the case of the LTE and the LTE Advanced technologies that will be widespreadly available in the next future. Another research direction is also possible, exploiting the idea of cooperation to improve the throughput offered to the user without having to change the network infrastructure. The Network Coding (NC) has been developed looking at the mobile wireless network limitations and taking account of the cooperation possibilities offered by dense environments. The goal is to propose a new routing scheme to improve the performance of routed wireless networks.

Thesis Contributions and Main Structure

This thesis is realized with the aim to go beyond the limits characterizing the actual available mobile communication systems. It aims to provide both theoretical and experimental contributions, presenting the design of a new NC-based coding algorithm and the validation in both numerically simulated- and real environments by experimental tests based on the Android Nexus 4 devices. The main characteristic of the proposed algorithm is the implementation ease: this means that the proposed algorithm - contrary to the others present in literature that require a lot of computational resources - can be implemented into small devices, like motes, since it will not require high computational power.

To allow a real-device implementation, new protocols and new structures are needed. In this thesis we present a complete environment, build over standard UDP/IP and TCP/IP links, with custom defined protocols and packets. All this definitions are used to develop an Android application that is used for real environment experimental tests on Nexus 4 smart-phones.

The real-case test scenario is a cooperative download of a common content by a number of personal devices. Each device runs the proposed application to simultaneously download some segments of the required content from both the Internet connection of the device and a local sharing platform based on the proximity. The Android devices that will run the test application will use innovative technologies, like Wi-Fi Direct, to join the cooperative peer-to-peer sharing network where the proposed 2-by-2 XOR NC algorithm will be used to exchange data and speed up the download procedure.

The structure of this thesis with the main contents of each chapter are reported below:

Chapter 1 presents the NC approach and the difference between the routing procedure that the NC theory proposes and the classical forwarding scheme. Starting from an overview of the NC theory present up to now in literature, we propose an alternative algorithm, based on the XOR opera-

tions. Accordingly, we characterize the scenario of application and then we summarize the key features that we can exploit into our original algorithm. Focusing to the 2-by-2 XOR algorithm, we explain which are the benefits in using the proposed algorithm and which are the key differences with respect to the classic NC theory. The last section of this chapter is dedicated to the definition of a set of two algorithms that will be compared with the proposed 2-by-2 XOR later in this thesis.

Chapter 2 introduces the numerical simulation for the proposed algorithms. To proof the validity of the original 2-by-2 XOR algorithm, a reference scenario is defined. All the algorithms are then numerically simulated into the specified scenario, to justify the implementation on real devices. The numerical simulation will give us an upper bound to valuate the performance of the implementation on real devices.

Chapter 3 presents an Android based application, developed with the aim of proof the validity of the 2-by-2 XOR algorithm in real scenario. The application permits, using innovative technologies like Wi-Fi Direct, to a group of users to download a remote content using a proximity-based cooperative platform. In this chapter will be presented all the application features, but also the new protocols and packet structures specifically developed for this application. The results that will be collected from such an application, with three Nexus 4 smart-phones, will be compared with the ones obtained from the numerical simulations, to validate the algorithms in a real scenario.

The thesis concludes with the analysis of the results obtained from the experimental tests conducted and with a quick review of the possible future filed of application for the proposed algorithm.

Chapter 1

Network Coding

In this chapter we define the problem tackled in this thesis, the scenario of applications and the related assumptions. We, then, present an overview of the NC state of the art, that will be used in the rest of the thesis as basis to develop and test new NC implementations. After a quick review of the state of the art, in section 1.4 we will propose a set of three original algorithms based on the NC theory tailored to be used in the specific cloud scenario, described in the following section. The proposed algorithms will be then compared to each other, analyzing the performance of the different implementations.

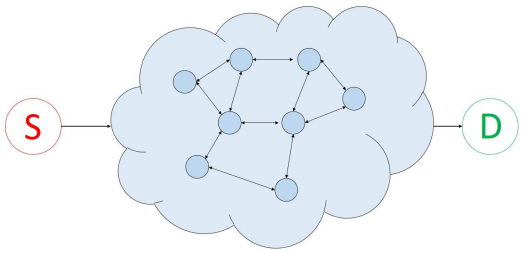


Figure 1.1: Single-source and single-destination wireless dense cloud network

1.1 Problem Formulation and Cloud Scenario

Consider a dense wireless peer-to-peer cloud meshed packet switching network, as in figure 1.1, with a single source S and a single destination D. The topology of the cloud network is unknown because the cloud has to be fully transparent for the source as well as for the destination node. In this scenario, every node that composes the meshed network is basically a router (for this reason known as routing node), that relays the incoming packets from source to the next hop, towards the destination. In a standard environment the network follows the *Kirchhoff rule* which states that all incoming flows into a node will also leave the node untouched. The output bandwidth of a routing node is, then, exactly the sum of the bandwidth from the single input streams (see chapter 1.2). Differently from conventional relay systems, in a NC enabled environment, each message sent on an output link by any router is a function, or mixture, of messages that had earlier arrived on the input links, as illustrated in figure 1.2, in such a way that the transmitted messages can be unmixed (or decoded) at the final destination[1].

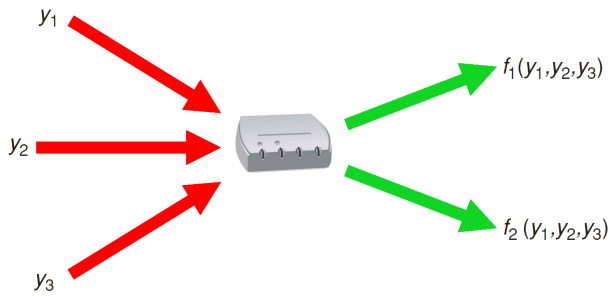


Figure 1.2: Router with NC functionalities

We define a set of three possible scenarios, depending on the number of information flows, source and destinations involved. The *unicast* scenario considered in the previous illustrative example, consists in a single information flow, with one source node and a single destination node. The *broadcast* scenario is an extension of the unicast scenario, with multiple different destination nodes for the same information flow, generated from a single source.

The most complex scenario is the *multicast* one, defined as multiple broadcast transmissions, with many information flows, generated respectively by many source nodes, each with multiple destination nodes.

In this thesis we focus on the broadcast scenario, where at a time a specific random node in the cloud network is the source of the information flow and the other nodes are the destinations.

Referring to the simplified scenario in figure 1.1, the problem we deal with is the propagation of data, from source to destination, through a dense wireless relay network, here referred as cloud network. In a cloud network relay and routing - traditionally performed at upper layers and centrally coordinated - are distributed over the nodes' physical layer, by means of NC techniques, making the cloud able to organize itself autonomously and adapt in critical dynamic scenarios. The goal of our work is to find a way to optimize the wireless medium usage to speed up the transmission in the cloud network, reducing the delay the receiver has to wait before the content is fully received.

The scenario is called *cloud* because the network between source and destination is completely transparent for the two source- and destination devices, and the nodes that belong to the cloud network do not require any type of a-priori knowledge about each other. As a consequence the cloud network is completely distributed without any coordination point and it is fully independent from the surrounding world.

The cloud scenario requires the nodes to use two different modalities to retrieve the data. They are called Remote Interface (RI) and Local intra-cloud Interface (LI) and are respectively a direct connection from every node to the remote source of the information (see figure 1.3), and a broadcast-based sharing platform with the neighbors that belong to the cloud network.

An illustrative example might be a conference room, where all the participants have to grab the teaching material from a common remote host. The scenario is composed by the two different activities, done by every device: as remote interface there is a direct packet switched connection between the

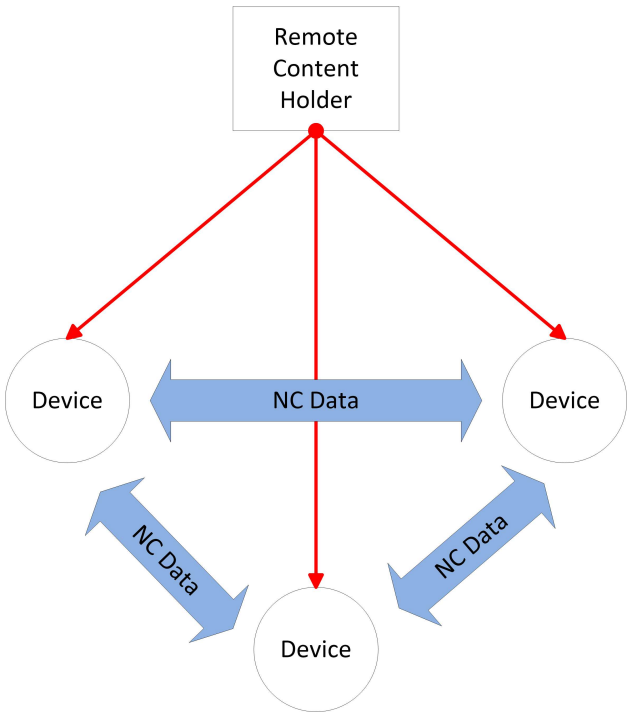


Figure 1.3: Schematic scenario representation, with the two Remote Interface and Local intra-cloud Interface links represented respectively in red and blue.

content holder and the device, while as Local intra-cloud Interface (LI) there is a wireless broadcast procedure to share information about the content with the cloud network. The aim of the LI usage is to minimize the amount of data transferred by the Remote Interface (RI) and maximize the global mean throughput of the cloud. In this second broadcast cooperation procedure we use the NC theory (see section 1.2) to optimize the wireless medium usage, speeding up the transmission between nodes.

In chapter 3 a demonstrator is proposed, which running on real personal mobile devices, is developed to validate the proposed methodology and to evaluate it in a real world environment.

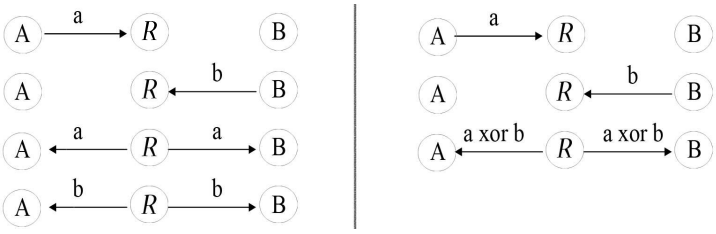


Figure 1.4: Simple comparison between classical forwarding and an NC enabled routing node.

1.2 Network Coding Theory

Mobile networks are nowadays an essential component of the modern life. They carry data to and from personal devices, like smartphones and laptops, but also carry data from sensors, remote control and many other applications. Wi-Fi networks, however, suffer from a severe throughput limitation and do not scale to dense large networks. The NC has been developed looking at this limitations and the goal is to propose a new routing scheme to improve the performance of routed wireless networks.

The NC advantages are many. The most important one is the network throughput improvement. Consider the fairly simple example, represented in figure 1.4, where two nodes A and B own a pair of packets to be exchanged, through a relay node R. what normally happens is that A sends its packet to the relay, which forwards to the destination, B. Then B does the same: it sends its packet to the relay which, this time, forwards it to A. As a consequence, each node obtains the right packet, otherwise this procedure needs four transmissions.

With a NC approach, we can save one transmission. The first transmission will be performed from A to the relay node, then from B to the relay R. At this point the relay can compute a XOR of the two packets and send the result back to node A and B, it sends the same packet at the same time. Node A can decode the packet sent by B applying another XOR operation between the received packet and the one it owns. The same can be done by node B to decode the packet sent by node A. This approach saves a transmis-

sion slot, useful for another information flow, and reduces the global delay the user has to wait to retrieve the information required, using in a better way the available radio resources. In fact the packet returned from the relay node will be:

$$a \oplus b \tag{1.1}$$

So, node A can do

$$(a \oplus b) \oplus a = (a \oplus a) \oplus b = 0 \oplus b = b \tag{1.2}$$

and correctly recover the packet sent by B. The advantage, in therms of time slots used by the transmission is 4/3, using such a simple NC scheme. NC techniques exploit the shared nature of the wireless medium, in fact any transmission is a broadcast one, in a spatial localized neighborhood.

There are other secondary benefits in using NC, such as the reduced energy per bit used by the devices, or like the lower delay in the delivery of the data to the destination (measured e.g. as the number of hops for a packet to reach the receiver).

There are many technologies that already use NC theory, like COPE (Coding OPportunistically) [2], that are new forwarding architectures that focus on the throughput improvement. The idea is to insert a coding scheme between the Internet Protocol (IP) and the Media Access Control (MAC) layers, allowing to encapsulate multiple packets, into a single transmission. This is done allowing the node to mix the messages it receives, in such a way that they can be unmixed at the destination. The goal is to use the wireless medium in a more efficient way, to guarantee more bandwidth to the devices with the same infrastructure.

The principles of NC are moderately simple. In a nutshell the idea is to allow the routing nodes to send as output a combination of the inputs. This means the router has to execute some operations to the payload of the incoming packets, in order to create a new packet to be transmitted. The key point is that the result of the combination process gives as output a packet with the same size of the incoming ones. This way it is possible to carry in

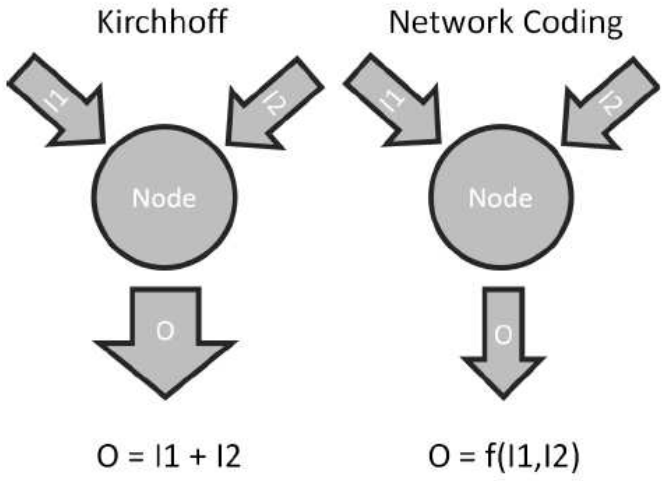


Figure 1.5: Kirchhoff network vs. NC.

a single transmission the information about all the packets incoming in the routing node.

There are many ways to put information about multiple incoming packets into a single one. Generally linear operations are used to keep the receiver simple. COPE [2], for example, uses XOR, while many other implementations use a generic linear combination of the packets, as shown in [3] and [15].

The difference between the classic relay approach, Kirchhoff, and the NC one, is that any router does not forward each received packet as it is, but it forwards a function, i.e a linear combination, of all the received packets, as shown in figure 1.5. As a consequence, in a NC environment, the data outgoing from the node are not the same that enter the node itself.

Time saving is not the only advantage in transmission coding: channel coding adds also redundancy to make the transmission over any medium more robust against errors.

Other coding techniques, like channel coding or source coding, are considered to be end-to-end techniques as they are applied on a single communication hop between two communication nodes, e.g. mobile device and base station. From a system point-of-view they are not limited to a specific layer,

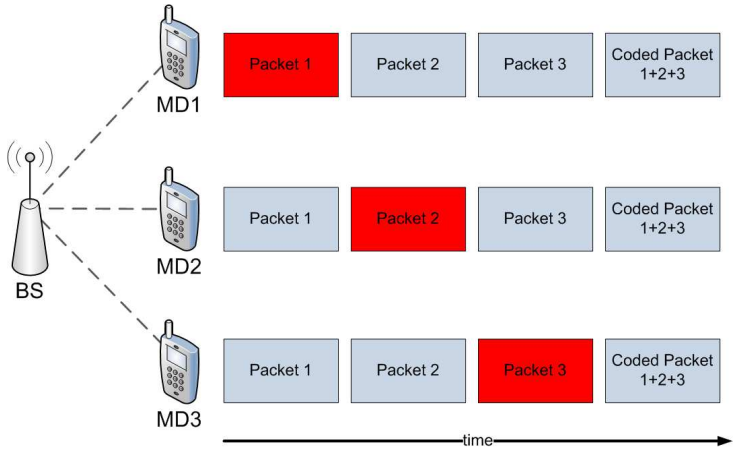


Figure 1.6: NC in broadcast scenario.

but could be used at the application-, network-, or physical layer. In contrast to source and channel coding, NC breaks with the end-to-end paradigm as it enables coding on the fly at each node in the communication network.

With NC the order in which the nodes send is important. If the relay node transmits a received packet right after reception, there is no coding possibility. To exploit NC, the relay needs to accumulate two or more packets, before it can start to send data.

1.3 Applications

NC theory has a large field of applications. Here is a list of the most interesting ones in terms of possible economical revenue, for commercial mobile platforms.

1.3.1 Wireless NC for error recover

Like in [25], we can use NC for error recovery purpose. Consider a typical broadcast scenario, composed by a central station - an Access Point (AP) or Base Station - that covers with his signal multiple devices. Every AP transmission is broadcast, because of the broadcasting nature of the physical

layer. The devices have a set of packets to be exchanged. In case every device looses a packet, i.e. because of a local interference that corrupts the received packet, there should be a recovery algorithm to request a new transmission of the packet. Consider the case where there are many devices, each loosing a random packet. There would be many re-transmissions, which are always broadcast, useful only for one device, or a little subset of devices. For all the other devices, that already own the retransmitted packet, the retransmission are useless; this problem is known as the *collector's problem.*[1]

In this case a NC scheme can be very useful. In figure 1.6 is represented an example with three devices. Every device sends a packet, the others will receive it. They are connected to the same base station. In this scenario we assume that each mobile device looses only one packet. Without NC, the error pattern over the devices would have an impact on the number of the retransmitted packets. If each device lost the same packet, the base station would have to transmit just one packet. In the worst case - where each device loses a different packet - three retransmissions would be needed. For the given example, NC requires transmission of only a single coded packet. In this case, the coded packet is a bitwise XOR combination of all packets. After receiving the coded packet each mobile device could recreate its missing one by combining the coded packet with the uncoded received packets.

1.3.2 Wireless NC in meshed networks

Let us consider a meshed scenario, with three mobile devices which are partially connected, i.e. two mobile devices that are out of range of each other and require the help of the third mobile device, as relay, to communicate. Figure 1.7 shows the exchange of two packets of the outer devices using simple relaying mechanism. This transmission procedure requires 4 time-slots. Figure 1.8 illustrates how a simple NC algorithm can improve the throughput in this basic scenario: the relay node sends back to the two devices the result of the XOR operation, applied to the two received packets. As a consequence

[1] inserire ref.

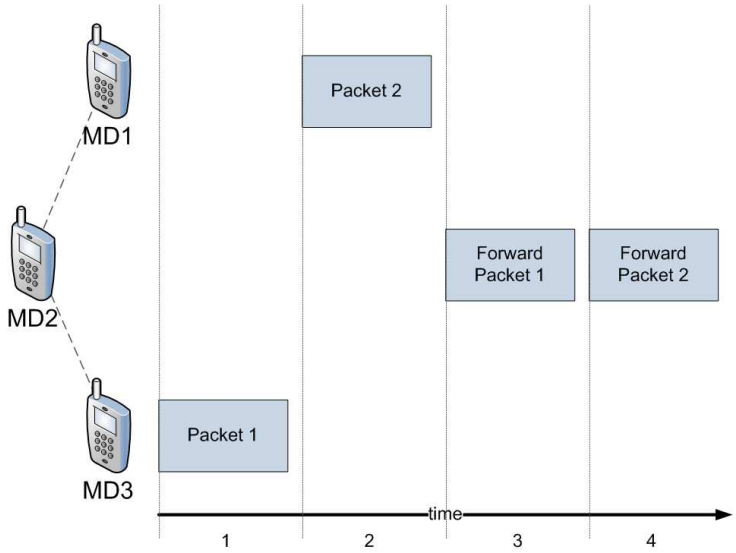


Figure 1.7: Mesh network, without NC.

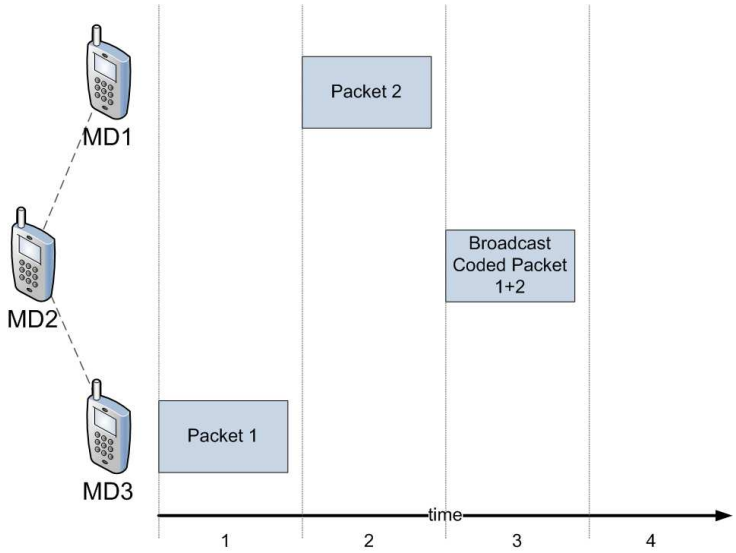


Figure 1.8: Mesh network, with NC.

every device can recover the missing packet, and the total transmission time is decreased by one. In IEEE 802.11, MAC assures that the capacity of the wireless channel is fairly shared among all mobile devices. In case that the outer devices send a large amount of packets, the relay node does not have enough capacity to support the relaying as it needs to send twice as much as the outer devices. NC is useful to ensure the relay sends the same amount of packets as the other devices.

Car Communication

There are many interesting projects in the car communication sphere, mainly based on [5]. They demonstrate how NC is suitable for meshed networks with dynamic topologies and intermittent connectivity, exactly the conditions in which people should work to bring mobile connectivity into cars. For example there are projects based on many access points, distributed on an highway, with many cars moving on it. Every car stays in range of a single access point for a very short time, so it is impossible to detect which packets are correctly received and which are not. In such a scenario NC is used to send new generated packets to the car, until it has enough information to decode the content.

Mobile Cooperation

The idea is to extend cellular links, as well in mobile phones as in base stations. By connecting to neighboring devices, called *cooperative clusters* are created, which guarantees to reduce energy costs, increase bandwidth and augment the robustness of the network. An example can be found in [6] and [7].
This is the idea, on which this thesis is based.

Mobile P2P Communication

The main idea is to spread information we find in a device to many others, using a local broadcast transmission. This is referred to as viral P2P communication. The advantage of NC in this context is that the source devices only need a minimal amount of knowledge about the targets received packets and therefore only a minimal amount of feedback is needed to ensure reliable data delivery.

1.4 NC Algorithms for Cloud Scenario

In this section we analyze three algorithms based on the NC theory, tailored to solve the specific problem. As introduced in section 1.1, the LI is the wireless interface that implements the NC scheme. The objective of this interface is to supply a platform where all the participants in the cloud network can share information about the content.

We assume that the content is divided into packets called *chunks*, inspiring from the infrastructure proposed in [12]. The idea is that all the devices already own some chunks, that were acquired from the RI, but they are not sufficient to decode the whole content. The devices have, therefore, to exchange chunks, with some rules and some protocols, in order to acquire enough information to decode the content. The goal is to minimize the data exchanged, the time the process needs and the power used during the execution.

In order to realize the cooperation procedures we consider three algorithms.

1.4.1 Simple Local Broadcast

This algorithm is the simplest one. Every device, when enabled to transmit, chooses one of the local saved chunk and transmits it, as it is, to the other devices. Any other device, that is overhearing the channel, can retrieve it

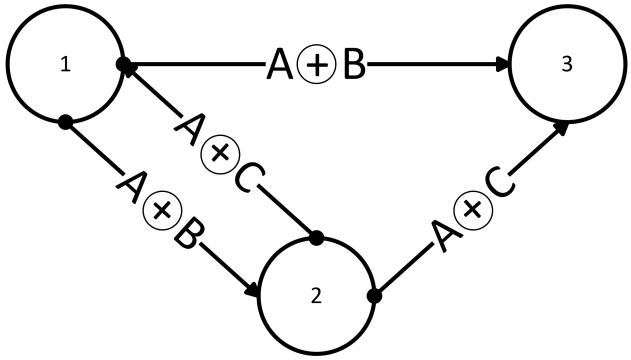


Figure 1.9: Three nodes local distribution system

and save it. The node only sends a header, indicating the index of the chunk transmitted, followed by the chunk data. This solution is agnostic about any network parameter and device status: it does not require any cooperation between the devices but it does not guarantee any type of performance.
This algorithm is used as a reference for the others.

1.4.2 2-by-2 XOR "Bit-Torrent"

This algorithm is not present in literature, it was developed within this thesis, using a BitTorrent-like approach. The goal of this algorithm is to achieve results comparable to the more complex NC pedantic implementation, called Global Linear Combination (GLC) methodology. The key point of this new method is the simplicity. In comparison to the GLC algorithm, this one is much simpler and can be adopted even in scenarios where the devices does not offer much computational power to execute complex finite field operations. This algorithm extends the COPE [2] idea about the use of XOR in transmissions. Because of the XOR operation, we need to select two chunks to be transmitted, XORed together. At every chunk transmission, the node has to send a header, containing the index of the two chunks transmitted, followed by their XOR. Assuming that this rule is used at every transmission, the client has to find the best combination of chunks that maximizes the profit of the transmission. Here we propose to use the most common and the most

rare chunk in the network. The most common chunk is defined as the chunk that is owned by most of the devices. The most rare is defined as the chunk that is missing by most of the devices. This particular metric is adopted with the aim of the transmission benefit maximization; as a consequence, using this rule we can maximize the number of devices that can obtain useful information from the transmission. To do this, the client needs a list of all the chunks owned by all the other clients, thus some information traffic is necessary, to periodically inform the rest of the network about the chunks every device owns. This information is exchanged periodically by all the nodes, at a schedule that varies with the speed of the network on which the devices belong: we need an accurate chunk distribution list, but the overhead should not be significant. A trade-off is necessary: we need to select the right time the node waits until it transmits its list of owning/missing chunks.

An example can be found in figure 1.9. We suppose, then, the network is composed by only three nodes. The node 1 owns chunks A and B, while the node 2 owns chunks A and C, the node 3 owns only chunk B. The first node with permission to transmit, is node 1, that transmits the packet $S = A \oplus B$, the only one he owns. Now, node 2 and node 3 can recover useful information doing respectively $B = S \oplus A$ and $A = S \oplus B$. Subsequently, node 2 has the chance to transmit and choses chunks A and C, because of the rule of the most common and the most rare chunk. Consequently node 2 transmits $S = A \oplus C$. The node 1 decodes $C = S \oplus A$ and the node 3 makes the same, $C = S \oplus A$.

1.4.3 Global Linear Combination

This is the pedantic implementation of the theoretical NC scheme, inspired by [11] and by the model proposed in [19], here considered as benchmark for performance evaluation. This algorithm is based on a particular NC implementation called *Random NC*, exposed in [10] and [13]. At every transmission, the node performs the encoding procedure showed in following section. The result is a so called *encoding vector* $\mathbf{g}$ the data segment $\mathbf{x}$, they are to

be transmitted together in a single packet.

Supposing this is the *kth* correct reception for the node, every active node that receives the transmission saves the data segment as $\mathbf{x}_k$ and the relative encoding vector as $\mathbf{g}_k$. Since all the nodes have information about the length of the content, composed by n chunks, the node can evaluate the download progression and terminate the procedure when it is finished.

Encoding

Assuming that we have n uncoded packets $\{\mathbf{m}_1, ..., \mathbf{m}_n\}$, defined in the finite field $\mathbb{F}$, generated from the same source S. In linear NC each packet $\mathbf{x}$ propagated through the network is associated with a vector of coefficients $\mathbf{g} = [g_1, ..., g_n]$. All the coefficients and the packets are defined in the finite field $\mathbb{F}$. The packet $\mathbf{x}$ is defined as:

$$\mathbf{x} = \sum_{i=1}^{n} g_i \cdot \mathbf{m}_i \tag{1.3}$$

The summation has to be done at every position in the symbol, so, assuming that $k = 1, .., s$ we get:

$$x_k = \sum_{i=1}^{n} g_i \cdot m_{i,k} \tag{1.4}$$

where $m_{i,k}$, x_k are the elements in position k of $\mathbf{m}_i$ and $\mathbf{x}$ respectively.

The resulting packet to be sent will contain both the *information vector*, namely the encoded data $\mathbf{x}$ defined in (1.3), and the *encoding vector*, that is the coefficient array $\mathbf{g}$. The encoding vector contains useful information that will be used by the receiver node to decode the content when the transfer is completed. Without the encoding vector, indeed, the receiver will be not able to successfully decode the information.

The encoding vector carries information about the indexes of the packets which are encoded into the transmission; for example, the encoding vector $\mathbf{g}_i = [0, ..., 0, 1, 0, ...0]$, where 1 is at the *ith* position, means that the information vector is equal to the $\mathbf{x}_i$ original packet, that means it is not encoded.

It is possible to encode a previously encoded packet, in the same way we encode an original uncoded packet. As a consequence, it is possible to perform the encoding process recursively. We will consider a node that has received and stored a set $(\mathbf{g}_1, \mathbf{x}_1), ..., (\mathbf{g}_m, \mathbf{x}_m)$ of m encoded packets, where $\mathbf{g}_i = [g_{i,1}, ..., g_{i,n}]$ is the encoding vector associated to the *ith* packet $\mathbf{x}_j$. This node may generate a new encoded packet $(\mathbf{g}', \mathbf{x}')$ by picking a set of coefficients $h_1, ..., h_m$ and computing the linear combination. The corresponding encoding vector $\mathbf{g}'$ is not simply equal to $\mathbf{h}$, since the coefficients are with respect to the original packets $\mathbf{m}_1, ..., \mathbf{m}_n$; in contrast, straightforward algebra shows that it is given by:

$$\mathbf{x}' = \sum_{i=1}^{m} h_i \cdot \mathbf{x}_i = \sum_{i=1}^{m} h_i \cdot \sum_{j=1}^{m} g_{i,j} \cdot \mathbf{m}_j \tag{1.5}$$

Considering

$$g'_i = \sum_{j=1}^{m} h_j \cdot g_{i,j} \tag{1.6}$$

The result is:

$$\mathbf{g}' = [g'_1, ..., g'_n] \tag{1.7}$$

where $g_{i,j}$ is as usual the *jth* element of array $\mathbf{g}_i$. The node will transmit $\mathbf{g}'$ and $\mathbf{x}'$.

Decoding

We will consider a node has received the set $(\mathbf{g}_1, \mathbf{X}_1), ..., (\mathbf{g}_m, \mathbf{X}_m)$ of m encoded packets. In order to decode the original uncoded packets, the node has to solve the linear system composed by m equations:

$$\begin{cases} \mathbf{X}_1 = \sum_{i=1}^{n} g_{1,i} \cdot \mathbf{M}_i \\ \vdots \\ \mathbf{X}_m = \sum_{i=1}^{n} g_{m,i} \cdot \mathbf{M}_i \end{cases} \tag{1.8}$$

Where $\mathbf{g}_k = [g_{k,1}, ..., g_{k,n}]$ and $g_{k,i}$ is the *ith* element of the $\mathbf{g}_k$ vector. This means:

$$\begin{cases} \mathbf{x}_1 = \mathbf{g}_1 \cdot \mathbf{M} \\ \quad \vdots \\ \mathbf{x}_m = \mathbf{g}_m \cdot \mathbf{M} \end{cases} \tag{1.9}$$

The general model is:

$$\mathbf{X} = \mathbf{G} \cdot \mathbf{M} \tag{1.10}$$

where the $\mathbf{G}$ matrix is composed by all the $\mathbf{g}$ row vectors:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_n \end{bmatrix} \tag{1.11}$$

and the $\mathbf{X}$ matrix is composed by all the $\mathbf{x}$ row vectors:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \tag{1.12}$$

In the same way, the matrix $\mathbf{M}$ is composed by all the $\mathbf{m}$ row vectors:

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix} \tag{1.13}$$

In this representation, unknowns are M_i, so we have a system with n unknowns and k equations:

$$\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_k \end{bmatrix} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,n} \\ \vdots & \ddots & \vdots \\ g_{k,1} & \cdots & g_{k,n} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_k \end{bmatrix} \tag{1.14}$$

that can be reversed in:

$$\begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_k \end{bmatrix} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,n} \\ \vdots & \ddots & \vdots \\ g_{k,1} & \cdots & g_{k,n} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_k \end{bmatrix} \tag{1.15}$$

that is, in compact vectorial form:

$$\mathbf{M} = \mathbf{G}^{-1} \cdot \mathbf{X} \tag{1.16}$$

We need at least $k \geq n$ to solve the system, that means we need to store a number of received packets (k) that are equal or greater than the number of the packet to be decoded (n). It might happen that not all the equations are linear independent, so the condition $k \geq n$ is not sufficient. Moreover, we need to check for the existence of the inverse of the $\mathbf{G}$ matrix that gives us another, more strict, constraint: $rank(\mathbf{G}) = n$.

Numerical example:
We now suppose to have three devices, and the content is:

$$\mathbf{M} = \begin{cases} m_1 = 6 \\ m_2 = 7 \\ m_3 = 3 \end{cases} \tag{1.17}$$

As explained before, the general model is:

$$\mathbf{X} = \mathbf{G} \cdot \mathbf{M} \tag{1.18}$$

We suppose the first device owns the second and the third chunk: m_2 and m_3. It will broadcast to the second and third devices the following:

$$\mathbf{g}_2 = [0, 3, 4] \tag{1.19}$$

and

$$x_2 = 6 \cdot 0 + 7 \cdot 3 + 4 \cdot 4 = 33 \tag{1.20}$$

The second device already owns the first and the second chunk, it saves the received bytes and broadcasts the summation. We suppose he chooses $g_1 = 2$ and $g_2 = 4$, so:

$$\mathbf{G}_3 = [2 + 0, 4 + 3, 4 + 0] = [2, 7, 4] \tag{1.21}$$

and

$$x_3 = 33 + 2 \cdot 6 + 4 \cdot 7 = 73 \tag{1.22}$$

Thus, the last device, who already owns just the first chunk, has:

$$\mathbf{g} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 3 & 4 \\ 2 & 7 & 4 \end{bmatrix} \tag{1.23}$$

and

$$\mathbf{X} = \begin{bmatrix} 6 \\ 33 \\ 73 \end{bmatrix} \tag{1.24}$$

The rank of the $\mathbf{G}$ matrix is 3, equal to the number of the rows in $\mathbf{M}$, so the device can recover the original data doing:

$$\mathbf{M} = \mathbf{G}^{-1} \cdot \mathbf{X} = \begin{bmatrix} 6 \\ 7 \\ 3 \end{bmatrix} \tag{1.25}$$

Chapter 2

NC Performance Analysis for Simulated Scenario

In this chapter we present numerical simulation results obtained by a Matlab numerical simulator, to show what could be the practical advantages in using the NC theory. They will be also used as a reference for the validation by the demonstrator based on mobile devices. Some limitations are present into the simulated scenario (see section 2.2.6) to simplify the implementation of the theoretical scenario. Thus the numerical results are considered as an upper bound for the performance that the real application can not exceed.

2.1 Scenario of Simulation

The scenario of the simulation is slightly different from the theoretical one described in section 1.1. The main difference is the topology used for numerical simulations.

2.1.1 Description of Scenario

During the numerical simulations we assume that many nearby devices need to access the same remote content. The content is itself divided into many segments (chunks), each of the same length. The last assumption is that

the devices that belong to the cloud network are all in full visibility, meaning that there is no need of relay nodes between the devices in the cloud network.

Under this assumptions, the interfaces described in section 1.1 are more specifically defined, tailored for the simulation purpose. The two interfaces work together simultaneously exchanging packets each with its own remote party, with the goal of the throughput maximization.

2.1.2 Interface definition

Remote Interface (RI)

The RI, as you can see in figure 2.1, consists in a direct transport link between the content holder, the host, and the device. This is the representation of a mobile data service provider, serving packet connectivity at every device. This interface is characterized by slow speed and medium packet loss probability.

The goal is to minimize the amount of data received by this interface, because of the metered nature of this service, of the slowness and, last but not least, of the energy usage of this interface usually larger with respect to the LI defined in the following section.

Local Intra-Cloud Interface (LI)

LI, represented in figure 2.2, acts as a sharing platform, where every device makes available to the others in the network its downloaded chunks. This interface enables direct peer to peer communication between all the devices, without any centralized coordinator. LI is characterized by high data rate and low packet loss probability, with respect to RI. The NC algorithms, described in section 1.4, are logically implemented in this interface.

2.1.3 Simulation Parameters

The Matlab numeric simulation is designed to allow some important parameters to be easily tuned by the user, as the **number of devices** involved

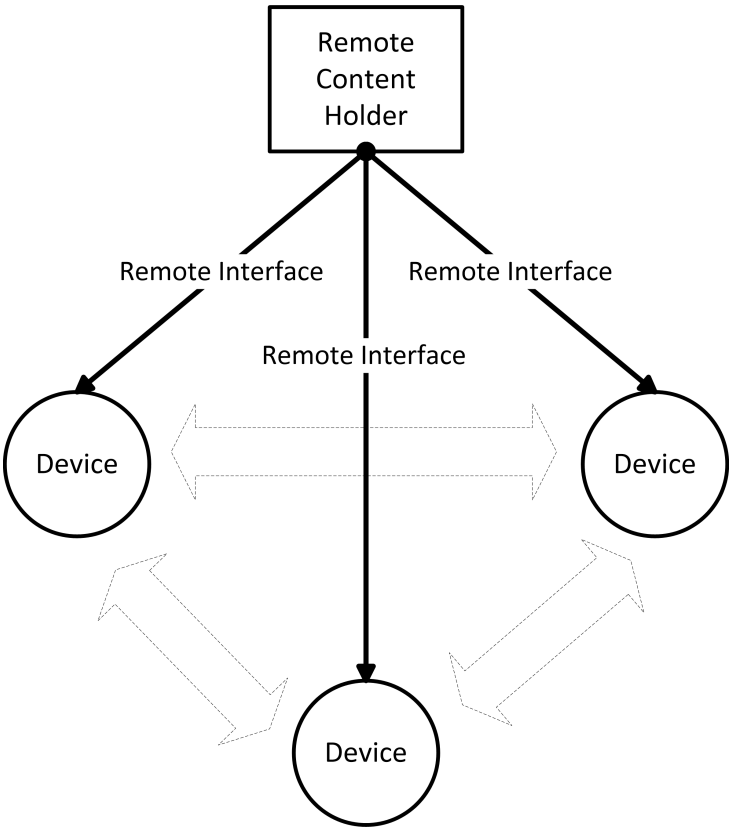


Figure 2.1: Remote Interface

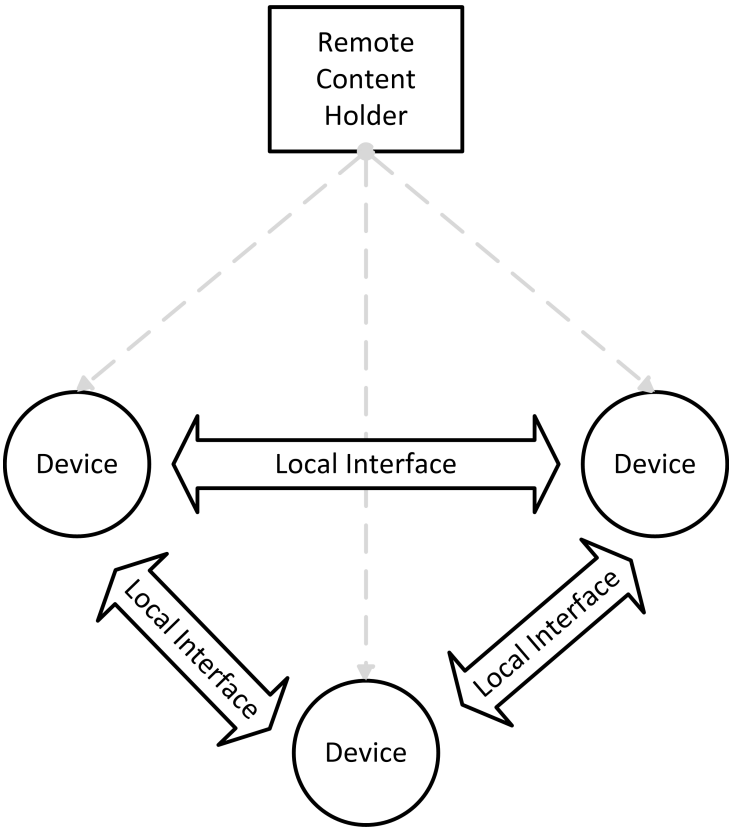


Figure 2.2: Local intra-cloud Interface

in the simulation, or the **number of run** the simulator has to evaluate for each set of parameter to complete the simulation. The output given by the numerical simulation is the mean value of the result of each run; this is useful to increase the precision. It is also possible to modify the **chunk size**, allowing the simulation of the Maximum Transmission Unit (MTU) of different network types. The interfaces are configured with two parameters, the link speed (expressed in kbps) and the loss probability, which is the probability that a packet sent from the source does not reach the destination. Thus, all network types can be simulated, varying this two parameters on both of the RI and the LI. In table 2.1 are collected typical values to correctly configure the interfaces. The represented values comes from many tests done by us during the development at the simulation environment. We will use exactly this values to get the results exposed in section 2.3. The user has to select also the content to be transmitted: there are no limitations on the file format or size, but to speed up the simulation process file size less of 1 MB is recommended.

Table 2.1: Common parameters used to simulate different transmission technologies.

Common Speed and Loss Probability parameters		
Technology	Speed	Loss Prob.
Wi-Fi 802.11 B	5,5 Mbps	2%
Wi-Fi 802.11 G	27 Mbps	1%
Wi-Fi 802.11 N	150 Mbps	1%
GPRS	50 Kbps	13%
UMTS	370 Kbps	7%
HSPA	6 Mbps	3%
LTE	50 Mbps	1%

2.2 Numerical Simulator Implementation

We have developed in Matlab code a NC simulator, with algorithms as plug-in. This means that it is possible to add or remove NC schemes, only adding or removing a simple Matlab code-file, without having to modify the core of the simulator itself. This makes very easy to test new algorithms.

The first step is to choose the chunk size, L, in bytes. This will affect all the numerical simulator parameters, so it is very important to choose it correctly. Let the content be in matrix $\mathbf{M}$, L bytes long. If the length is not exactly a multiple of L it is tail-padded with zeros. This way we have a matrix $\mathbf{M}[n \times L]$ that is the same presented in section 1.4.3, with the contend divided into n different chunks.

Each device involved in numerical simulation is represented by two tables: the *information matrix* $\mathbf{X}[m \times L]$ and the *encoding matrix* $\mathbf{G}[m \times n]$, where m is the number of the received chunks. The simulation process will fill row-by-row the $\mathbf{X}$ and the $\mathbf{G}$ matrix, following the rule of the selected algorithm, as if they were the memory of the devices. The matrices, used later to decode the content, contains the data that comes from the two interfaces, RI and LI. A coordination function is necessary, to give write priority to one of the two: we have decided to give the priority to the RI.

The time quantization inside the numerical simulation environment let us know ho many clock cycle the procedure uses to complete. To have a temporal reference, we need to know how long last in real time one clock cycle. We define the Local Interface Transmission Time (LITT) as the basic time unit used by the simulator, this will be the real time duration of a single simulated clock cycle.

$$LITT = \frac{ChunkSize}{LocalSpeed} \tag{2.1}$$

The simulator has to calculate how many time-slots require the remote transmission of a chunk, called Remote Interface Transmission Time (RITT), with respect to the LI speed:

$$RITT = \left\lceil \frac{LocalSpeed}{RemoteSpeed} \right\rceil \tag{2.2}$$

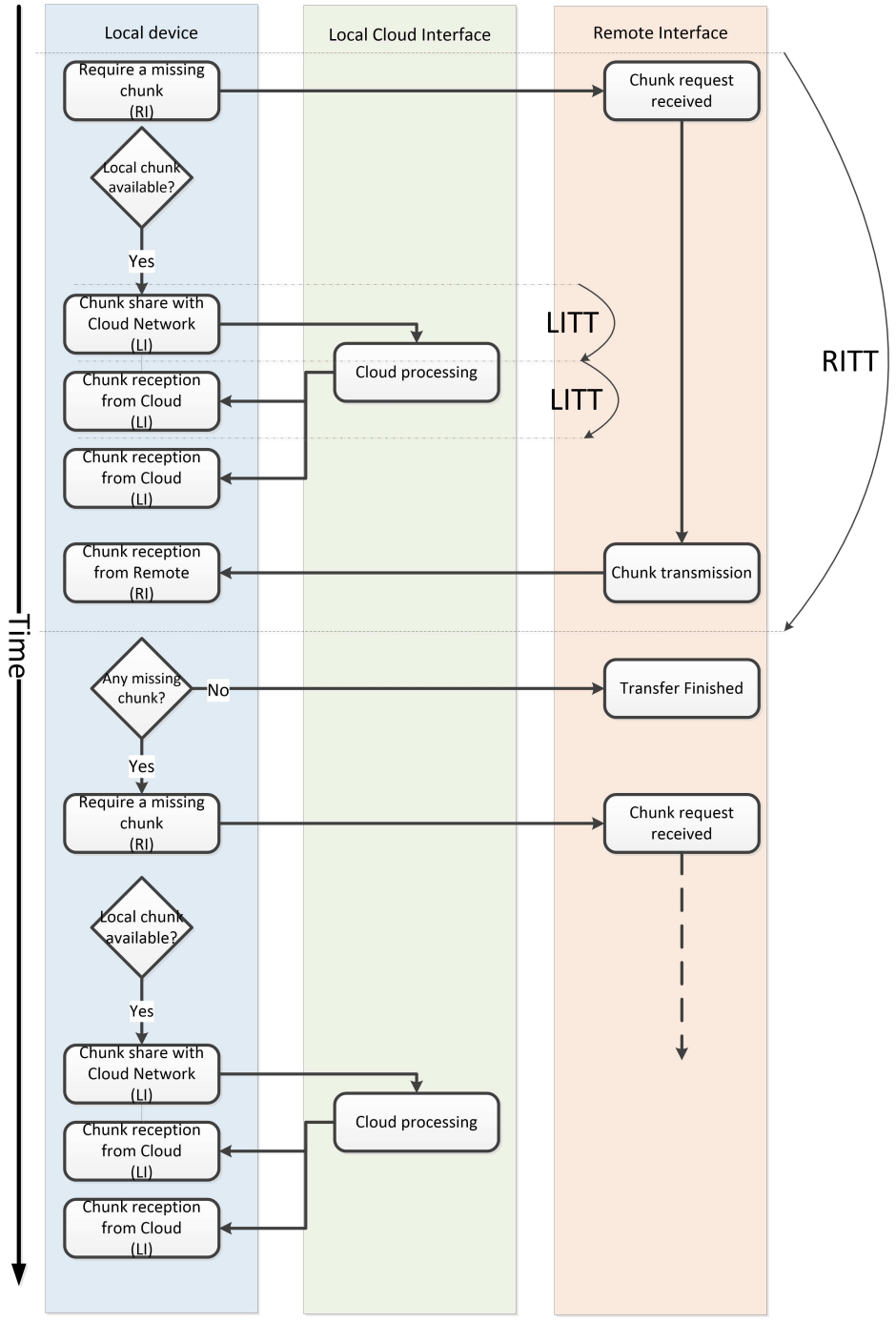


Figure 2.3: Schematic representation of the Global System.
LITT and RITT are respectively Local Interface Transmission Time and Remote Interface Transmission Time

The general idea used to develop the numerical simulator is represented in figure 2.3: at every clock cycle the program checks if it is time for a RI reception and in positive case starts the relative procedure, explained in the following section, to save the received chunk. In negative case, the RI reception phase is skipped. Afterward the local sharing procedure on the LI is started, to share or receive chunks from the cloud.

Every transmission, on both the RI and LI, has a reception failure probability: in numerical simulation this is represented with a test that returns a negative response with the interface specific probability. In case of negative response, the received packet is discarded by the specific device. Because of the test procedure is executed independently by every device, others might successfully receive the packet that has been discarded by another.

The download procedure is stopped into two different times, one for every interface. The RI is stopped when the device detects to have enough information to decode the content, i.e. when the $\mathbf{G}$ matrix is full rank. When the RI has been stopped, the LI still active until the completion confirmation from all the devices in the cloud network.

The parameters described in section 2.1.3, allow the numerical simulator to be a platform that allows the simulation of any scenario. Referring to the figure 2.3 it is easy to see how the parameters impacts on the structure. There are some parameters that can be varied without any consequence to the general scheme, such as the file size or the chunk size, as they only change the number of chunks available n and the length of the $\mathbf{M}$ matrix L respectively. Only the running time is affected to the variation of such a parameter. There are some other parameters that can also change the structure of the simulator itself. For example, when we simulate the RI trend alone, we disable the LI by setting the LI loss probability equal to 1. This way the structure is modified as in figure 2.4 and we can appreciate the By changing, as another example, the speeds, the system will only add or remove some LITT cycles into the RITT one. It is important to remark that the simulator is created assuming LI faster than RI, so the numerical

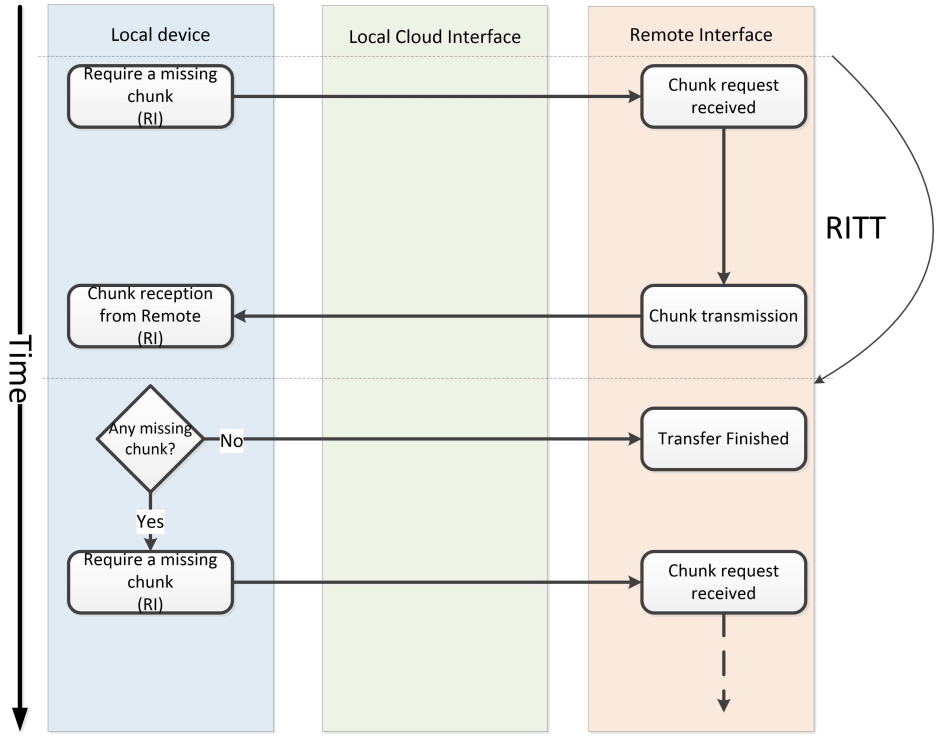


Figure 2.4: Schematic representation of the system where LI is disabled

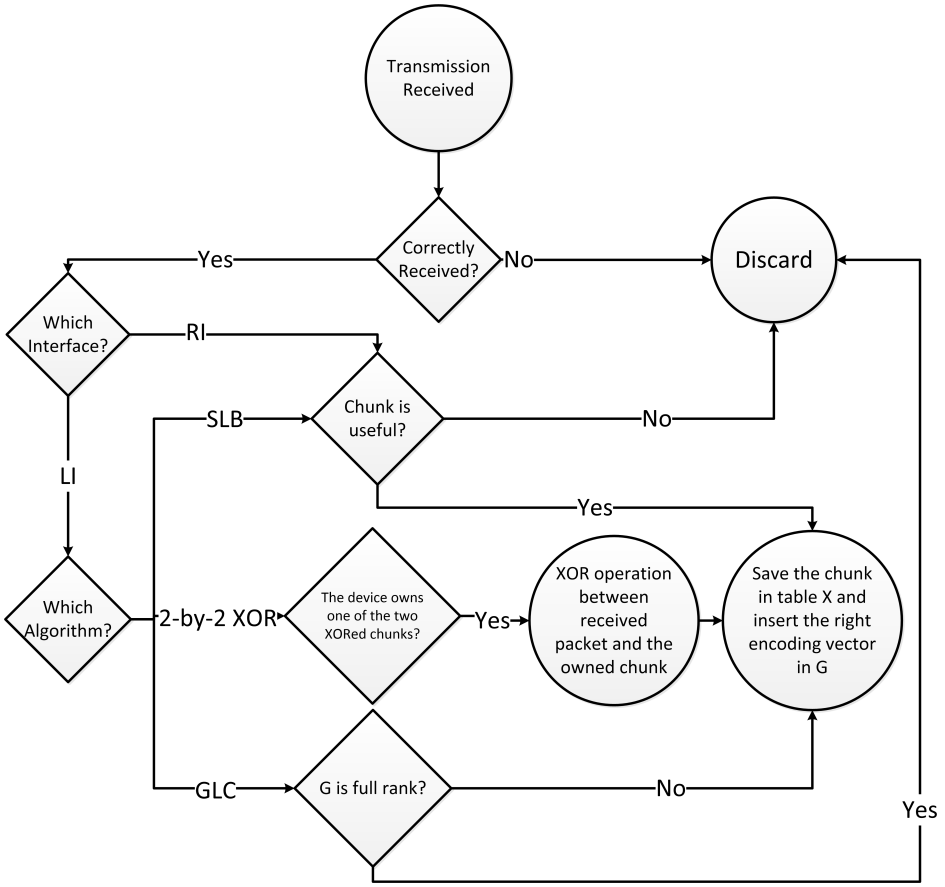


Figure 2.5: Schematic representation of the packet reception by a device in numerical simulated environment.

simulator checks the correct selection of the parameters by the user.

2.2.1 Packet reception

At every packet reception event, the system has to save the received information into the tables of the corresponding device. The packet reception procedure is invoked by each of the two available interfaces. The procedure acts differently, depending on the interface.

In case the procedure is called by the RI, that means we are ending an RITT. The remote download returns the index i of the received chunk, as well as the chunk itself $\mathbf{x}$. A new vector $\mathbf{g}$ is created, full filled of zeros, with

an 1 in position i, that is the index of the received chunk. This vector is inserted into the **G** matrix. Then the received chunk **x** is inserted into the **X** matrix.

When a packet is successfully received from the LI, the system gets two vectors, called **g** and **x** respectively. The following operations are done for every device. The first step is to check if the received information is useful for the device. This is done by inspecting the **G** matrix, checking if the rank increases after the insertion of the received **g** row. In case the rank increases, the data are useful and are saved. The system will insert a new row at the end of the **G** matrix with the *encoding vector* **g** received, consequently the corresponding information vector **x** is inserted at the same row index in matrix **X**.

There is also a service vector for every device, $\mathbf{r}[1 \times n]$, that is used only in the numerical implementation. This vector is not mandatory from the NC theory, and is used to track which chunks are known (we have received information about it) and which are not. It is initialized to zeros, when the node receipt a packet it will put an 1 at the index of the received chunks. In case the packet contains more than one chunk, all the . This is very useful to avoid the **G** matrix scan at every time-slot. This procedure is used to save data from both of the interfaces, regardless of the NC techniques usage.

2.2.2 Simple Local Broadcast

The Random Local Broadcast (RLB) algorithm chooses, at every time-slot, a random chunk from the known ones to be sent. The chunk is sent uncoded to the other devices. The RLB algorithm produces as output an encoding vector $\mathbf{g}'$ composed as following:

$$\mathbf{g}' = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix} \tag{2.3}$$

with the 1 in the position i representing the index of the chunk chosen to be transmitted. Assuming that the matrix $\mathbf{M}$ is composed by n row vectors $\mathbf{m}$:

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix} \tag{2.4}$$

the information vector $\mathbf{x}'$ is the *ith* row of the $\mathbf{M}$ matrix:

$$\mathbf{x}' = \mathbf{M}_i \tag{2.5}$$

Every device that correctly receives this transmitted data checks for the usefulness of this data and, in positive case, stores it with the procedure described in section 2.2.1.

2.2.3 2-by-2 XOR

Differently from the RLB case, the 2-by-2 XOR algorithm chooses two chunks to be transmitted. After the selection, the two packages will be sent as the result of the XOR ($\oplus$) operation.

The two chunks that the algorithm selects, are chosen as the most common and the most rare in the entire environment. As explained in the first chapter, we define a new metric, that measures the availability of any chunk across the network. This is done inspecting the G tables of all the devices with the aim to find which chunks are owned by every device. As a consequence, we can determine which is the most common and which is the most rare chunk in the network. We will use this information as the chunk indexes to be used in XOR function. The aim of this particular selection is to maximize the number of devices that can take benefit from the transmission. As explained in section 2.2.6, this procedure is implemented in centralized mode into the numerical simulator but the implementation running on real devices is truly distributed across all the nodes.

The 2-by-2 XOR algorithm produces as output an encoding vector $\mathbf{g}'$ composed as following:

$$\mathbf{g}' = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & \cdots \end{bmatrix} \tag{2.6}$$

with the 1 in the positions i and j representing the indexes of the chunks chosen to be transmitted. The information vector $\mathbf{x}'$ is composed by the result of XOR operation between the *ith* and the *jth* rows of the $\mathbf{M}$ matrix:

$$\mathbf{x}' = \mathbf{M}_i \oplus \mathbf{M}_j \tag{2.7}$$

Every device that correctly receives this transmitted data can know which chunks are present in the packet watching the encoding vector $\mathbf{g}$. The node checks for the usefulness of this data, checking the presence in the local storage of one of the two XORed chunks. In case none or both are found, the packet is discarded. In other case, the node use again the XOR operator with the encoded packet and the one it owns, decoding the missing one. Then it stores the resulting chunk with the correct index, using the procedure described in section 2.2.1.

2.2.4 Global Linear Combination

The GLC[4] algorithm is the only one that does not choose any packet: with this algorithm every device is sending all the packet it owns, at every transmission time, as explained in section 1.4.3. The practical implementation of the algorithm is inspired by [13], in which the authors present their experiences with a P2P content distribution system that uses NC.

The algorithm extracts the *kth* row from the $\mathbf{X}$ and the corresponding $\mathbf{G}$ matrix in two vectors, $\mathbf{x}_k$ and $\mathbf{g}_k$ respectively. Then it generates a new random coefficient h_k uniformly distributed across the finite field $\mathbb{F} = GF(2^s)$ with $s = 16$ in our case, used to generate the new vectors

$$\mathbf{x}'_k = h_k \cdot \mathbf{x}_k \tag{2.8}$$

and

$$\mathbf{g}'_k = h_k \cdot \mathbf{g}_k \tag{2.9}$$

This has to be done for every of the m packet the device owns, summing the results:

$$\mathbf{x}' = \sum_{k=0}^{m} h_k \cdot \mathbf{x}_k \tag{2.10}$$

and

$$\mathbf{g}' = \sum_{k=0}^{m} h_k \cdot \mathbf{g}_k \tag{2.11}$$

All the operations are done in the $\mathbb{F}$ finite field.

Every device that correctly receives this transmitted data checks for the usefulness of this data and, in positive case, stores it with the procedure described in section 2.2.1.

2.2.5 Decoding

When the download algorithm is finished, the decoding procedure can be started. The RI determines when there is no chunk to be received, analyzing the rank of the **G** matrix. During the execution of the decoding procedure, the LI is still active, providing chunks to the neighbors, while the RI is torn down.

The decoding algorithm is executed by every device on the received data **X** and the corresponding encoding matrix **G**. The procedure, explained in section 1.4.3, consists in the inversion of the **G** matrix to solve the system in (1.16)

2.2.6 Simulation Scenario Limitations

To build the NC numerical simulator, we have to find a compromise between the simulation speed and the environment variables, so we have chosen a scenario with some limitations.

First of all the simulator assumes a perfect coordination between local transmission, it is considered as an ideal token ring network (*IEEE 802.5*[1]). In a such network, indeed, only one device can transmit at a specific time. The device allowed in transmission, owns a *token*, that is exchanged with another node in the network after the end of the packet transmitted. This way is impossible to have collisions. In the real implementation this is not true, as we will use a different network type.

[1] inserire ref!

The 2-by-2 XOR (see section 1.4.2) requires some control traffic between nodes, to exchange information about the chunk availability of each node. The numerical simulator neglects this traffic, allowing the nodes to access directly to the memory of the other nodes by reading their matrices. Thus, this signaling traffic does not contribute in channel usage. In the implementation for real devices, this information are exchanged as normal packets over the network and this overhead increases the real network usage. As a consequence, the numerical simulation only provides an upper bound value for real systems.

The simulated version of the GLC algorithm (as explained in section 1.4.3) is fully implemented, but in our numerical simulations we can not consider the time the device needs for the final matrix inversion. We cannot take account of this time because this operation is high resource expensive, even in simulated environment, and therefore it is hardware dependent: numerical simulations done on different machines will give completely different results. Thus, in case of a GLC implementation on real device, the performance will be affected by this heavy post-download processing required to decode the content. The complexity of this operation depends also on the size of the finite field. Indeed a large finite field guarantees larger space for every possible combination of chunk coefficients and this means better performance of the algorithm, but this also slows down every mathematical operation. In fact the complexity of a square matrix inversion with size n, in a finite field $\mathbb{F}_q$ is $\mathcal{O}(n^3 log^3 q)$, so we have to find the best trade off.

Regardless of the invertibility checks, the inversion of the $\mathbf{G}[m \times n]$ matrix requires that the matrix itself is full rank. Because of the possibility of non linear independent rows, maybe $m > n$. In this case the $\mathbf{G}$ matrix is not simply invertible, as the system is overdetermined. The solution of such an overdetermined system of equations is extremely computationally expensive and might be unsolvable by current available machines. This issue is present only in the GLC scheme, because the others stores only the information needed. The GLC scheme, indeed, stores every single transmission the node

can receive, without any control over the usability of the information. To overcome this issue we have decided to implement a check whenever a new transmission is received, to know if it is an useful transmission or not. The check is simple, if the new coefficient vector $\mathbf{g}$ increases the rank of the $\mathbf{G}$ matrix, the information in the transmission are useful, otherwise they are discarded. This way we are able to save only the useful information needed to decode the content. This approach has another good side-effect: it reduces the memory requirement for every simulation, up to the 50%. Another requirement for fast numerical simulation is the usage of small files as the content to be transmitted. This helps in keeping the dimensions of the matrices as small as possible, allowing the numerical solution in reasonable time.

2.3 Numeric Results

The simulations are done by varying one of the parameters at time, and observing how the algorithms react to the change. The purpose of the numerical simulation is not only to compare the encoding schemes, but also to determine which are the conditions where the algorithms give out the best performance. The benchmark step is divided into three parts, dedicated to the remote, the local and the global system, meant as the combination of the two interfaces. This is done to have an idea on how every interface interacts with the other and contribute to the global result.

The output value of the simulator is the so called *Mean Throughput*, which is defined as the mean of the transfer speed measured by the devices. The transfer speed is calculated by every device independently on the two interfaces and then summed. Thus we can obtain information about the two interfaces separately.

2.3.1 Parameters for Numeric Simulation

The configuration we have chosen is the same for all the numeric simulations, to be coherent in measurements. In the following table 2.2 you can find the values we use.

Table 2.2: The parameters we use to run the numerical simulator

Parameters	
File Size	$3,82$ KBytes
Number of Device	12
Number of run per simulation	30
Chunk Size	100 Bytes
RI Speed	0.8 Mbps
RI Loss Probability	3%
LI Speed	27 Mbps
LI Loss Probability	1%

2.3.2 Remote Interface (RI)

The first step is to numerically simulate the RI alone, without any type of local cooperation. The result is used as reference for the valuation of the other results. As there is no cooperation, the only meaningful test is the global mean throughput versus the RI loss probability, as in figure 2.6. We have noticed that, as expected, the performance linearly decrease starting from the nominal $0.8Mbps$, when there is no loss, approaching to zero, when the loss probability approaches to one.

2.3.3 Local Intra-Cloud Interface (LI)

The test configuration is changed, because the content to be retrieved is now pre-distributed across all the nodes in random manner. The procedure

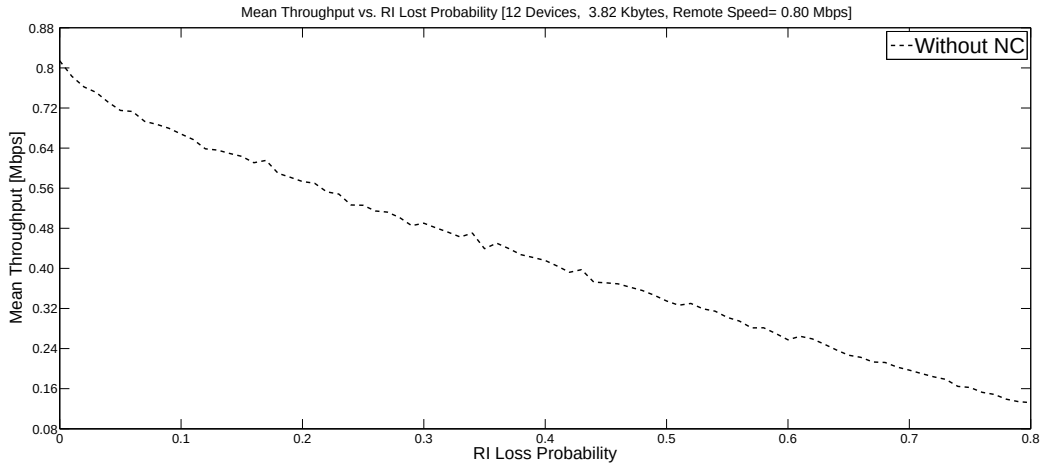


Figure 2.6: Global Mean Throughput vs. RI loss probability, in RI.

used to distribute the content across the nodes is designed to let every node to own a pre-defined quantity of random chunks. The chunk quantity is common for all the nodes and is regulated by the so called *Local Distribution Probability*, that is the probability for a chunk to be present at the node before the sharing procedure is started. To successfully complete the sharing procedure is necessary that at least one copy of every chunk is present in the network, otherwise the content cannot be decoded. Thus, a check after the chunk assignment is necessary, to correct invalid distributions.

Suppose that $\mathbf{M}$ is the content to be retrieved by a set of devices S, every node has a set of chunks $\mathbf{m}_s$ such that:

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_s \end{bmatrix} \tag{2.12}$$

The cardinality of $\mathbf{M}$, because of the possibility of duplicated chunks across many nodes, is:

$$|\mathbf{M}| = \sum_{k=1}^{S} |\mathbf{m}_k| - \sum_{i,j} |(\mathbf{m}_i \cap \mathbf{m}_j)|, \quad i \neq j \tag{2.13}$$

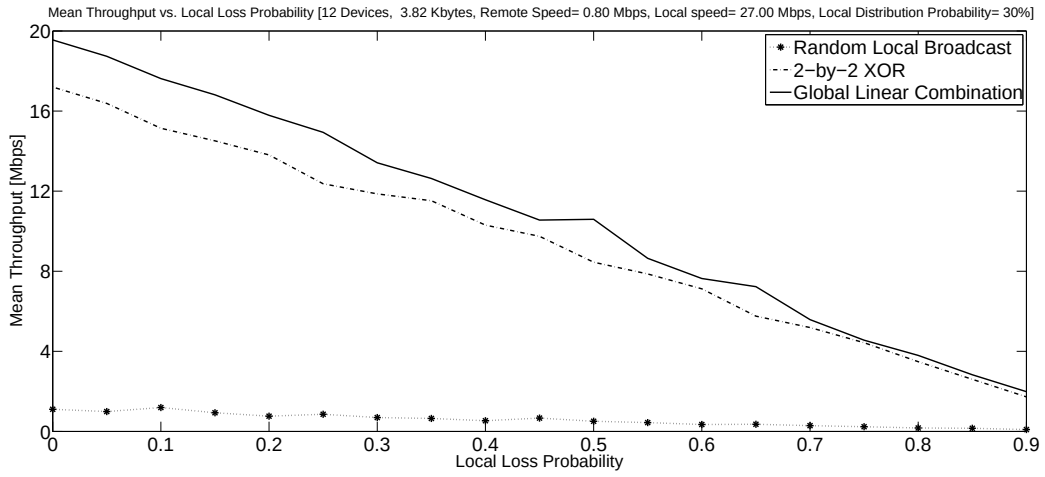


Figure 2.7: Global Mean Throughput vs. Local Loss, in LI.

As default we set the *Local Distribution Probability* to 30%, this means every node starts with the 30% of the chunks, randomly chosen.

In such a condition, a node by itself, cannot decode the entire content, but there is enough information in the network, to allow a cooperation algorithm to successfully share chunks and enable all the nodes to decode the content. The comparison between the three algorithms let us see which one is the most powerful by itself.

Throughput vs. LI Loss Probability

In figure 2.7 we have plotted the result of the simulation, where the variable parameter is the LI loss probability. Here we can demonstrate how every NC algorithm guarantees better performance in every loss condition with respect to the RLB reference. The two NC algorithms are comparable at high loss probability, but the GLC is roughly $\simeq 17\%$ faster than the 2-by-2 XOR at small loss probability. The 2-by-2 XOR algorithm guarantees ~ 14 times the throughput of the RLB algorithm, and the GLC extends the gap up to ~ 15 times.

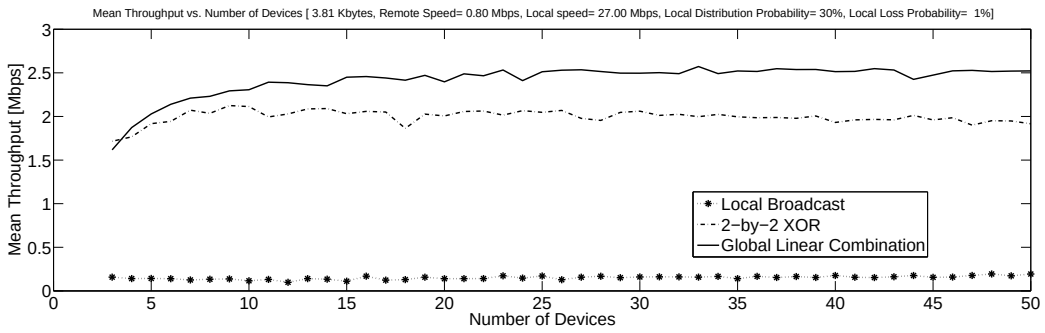


Figure 2.8: Global Mean Throughput vs. Number of Devices, in LI.

Throughput vs. Number of Devices

In figure 2.8 we have plotted the result of the simulation, where the varying parameter is the number of the devices. The larger is the number of devices involved, the larger is the benefit the devices can take from the cooperation, but a small set of 12 devices is enough to saturate the local network bandwidth (27 Mbps this case). A greater number of devices is useless to increase the global throughput, when using this network parameters. As before, the reference RLB marks a very poor result. Regardless of the number of device involved in the simulation the reference algorithm is 16 to 20 times slower the others: 2-by-2 XOR and GLC

Throughput vs. Local Distribution Probability

In figure 2.9 we have plotted the result of the simulation, where the variable parameter is the distribution probability of the chunks between nodes. This test measure the behavior of the algorithms for different overlapping chunks. We observe that, as usual, the difference is strong between the reference technique and the NC approach, but also that the most effective working state for all the two NC algorithms is around the 30% of overlapped information.

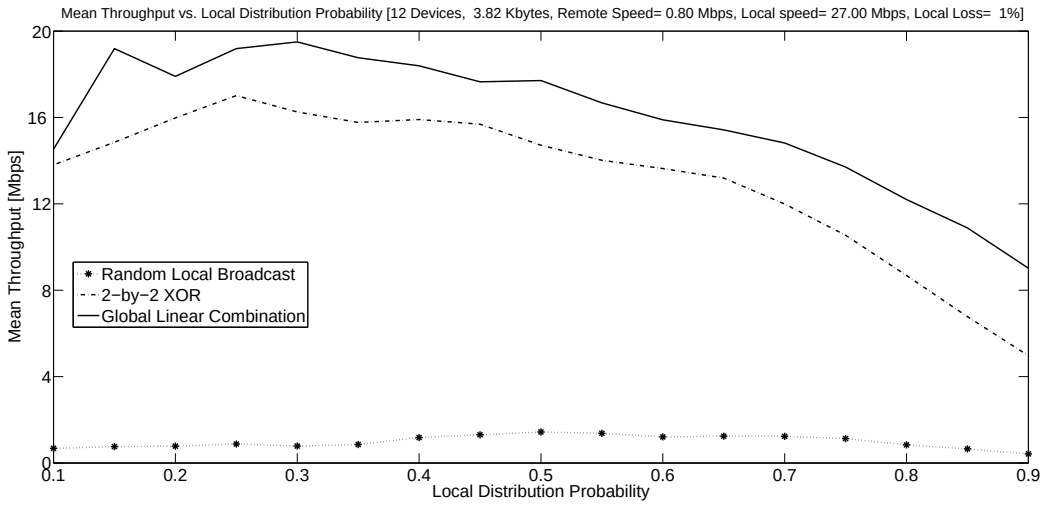


Figure 2.9: Global Mean Throughput vs. Local Distribution Probability, in LI.

2.3.4 Global Results

After all the single-interface numeric simulations, now we have enabled all the two simulated interfaces the same time, to simulate the entire scenario. All the configurations are the same as before.

Throughput vs. LI Loss Probability

In figure 2.10 is represented the relationship between the global mean throughput and the LI loss probability. In perfect no-loss case, the amount of traffic served by the combination of the two interfaces reaches the 7.2 Mbps value, for the GLC algorithm, and to the 6 Mbps value for the 2-by-2 XOR algorithm. The performance of the two NC enabled schemes are roughly 2 and 2.5 times the reference RLB one, but 9 and 7.5 times faster than the linear remote download.

As expected, when the LI loss probability approaches to 1, the mean throughput approaches to the 0.8 Mbps bound, which is the RI speed. In such a situation, indeed, the system completely disables local cooperation.

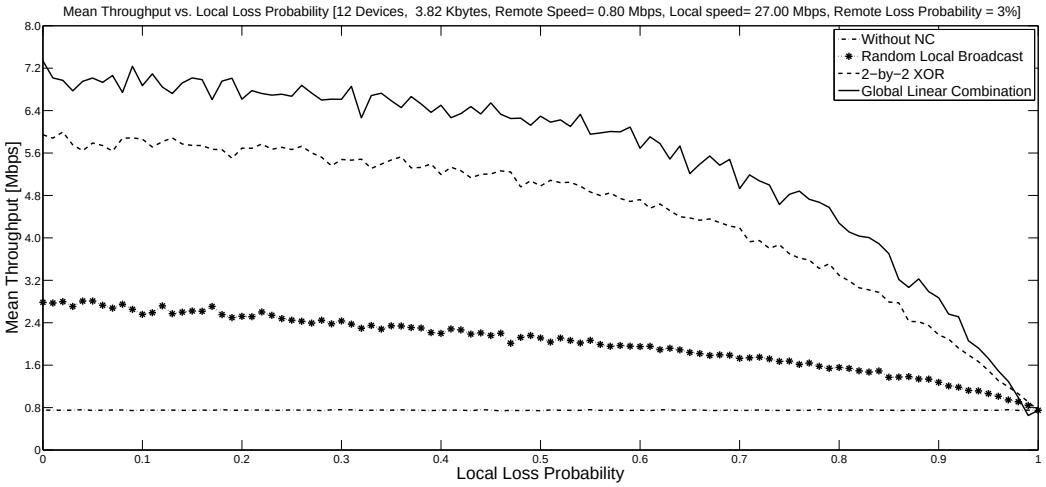


Figure 2.10: Mean Global Throughput vs. LI Loss Probability (Global system benchmark)

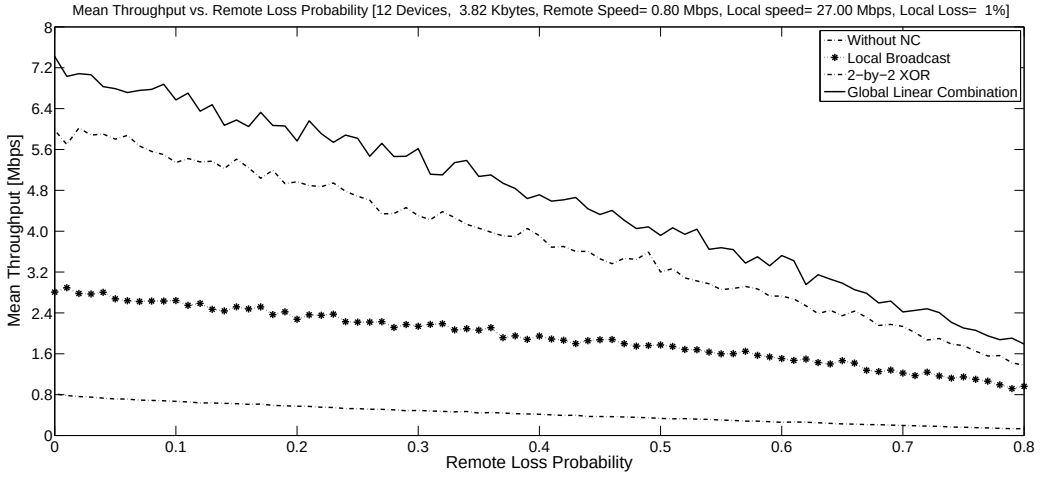


Figure 2.11: Mean Global Throughput vs. RI Loss Probability (Global system benchmark)

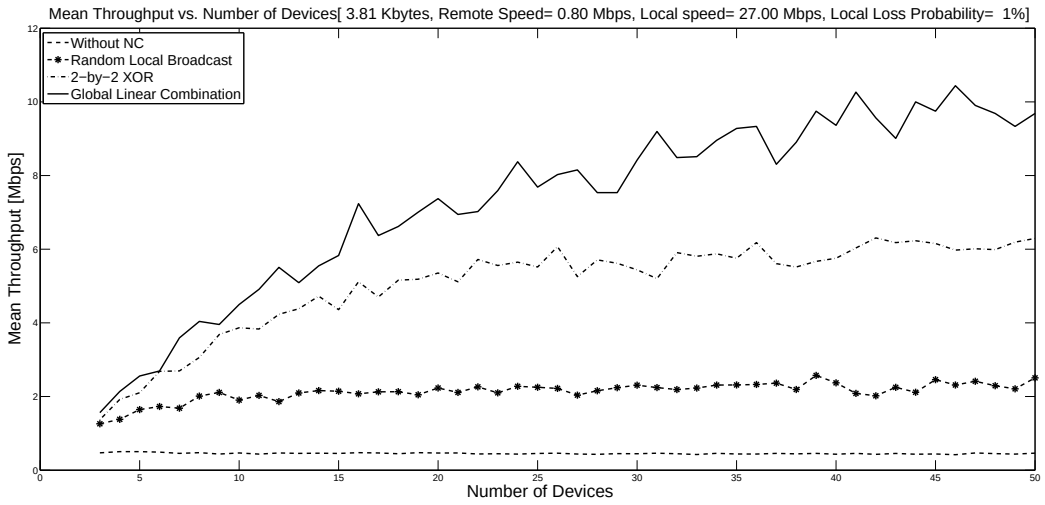


Figure 2.12: Mean Global Throughput vs. Number of Devices (Global system benchmark)

Throughput vs. Remote Loss

In the figure 2.11 we have a fixed local loss probability of 1%. In this case is more evident the relationship between the information flow incoming the entire system and the mean global throughput, flowing between the devices. The two NC algorithms are always 2 to 2.5 time faster than the RLB reference algorithm, and from 7.5 up to 9 time faster than the linear remote download, but the performance decreases linearly when the RI loss probability grows, because of the leak of the less information available in the system.

Throughput vs. Number of Devices

In the figure 2.12 we have plotted the evolution of the mean global throughput of the system, in relationship with the number of device involved in the cloud network. Is possible to see how a larger number of devices can improve the performance, up to the upper-bound represented by the LI capacity. In a dense environment is supposed to work ever in the local network saturation zone, because of the large number of devices involved. To limit the computational power needed by the simulations we have chosen the number of

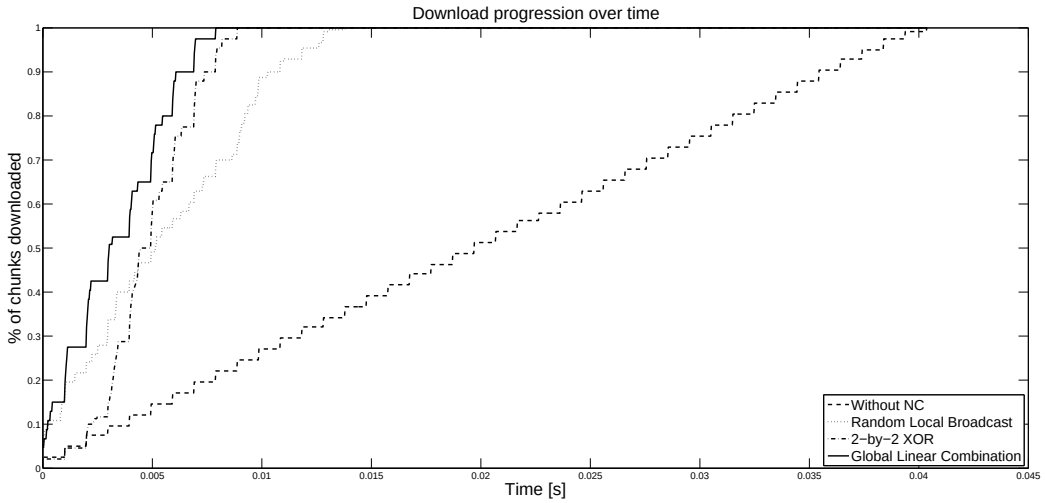


Figure 2.13: Average file completion evolution during time over all the nodes

12 devices, which is the slowest number of devices to saturate the 27 Mbps local network connection. Every NC enabled algorithm take advantage from a global major information in the system. As expected for 12 devices we can found the same values, $7.2Mbps$ and $6Mbps$ we have seen in the figure 2.11 and 2.10.

Download Progression vs. Time

The simulated model gives us many information about throughput of the system in many situations, but we have no information on how this speed evolves during time. For this purpose we have done another benchmark (figure 2.13) where the download progression of the download is plotted against the temporal evolution, for each algorithm. In figure 2.13 is possible to see how the RLB reference algorithm, as the GLC one, is starting immediately, marking an initial gap with the 2-by-2 XOR. The reason why the 2-by-2 XOR algorithm initially gives no help is that is necessary that a node owns at least two chunks, to successfully execute the algorithm. After the initial phase, however, the 2-by-2 XOR algorithm grows up faster than the others, returning at a lever comparable with the complete implementation of the NC theory (GLC).

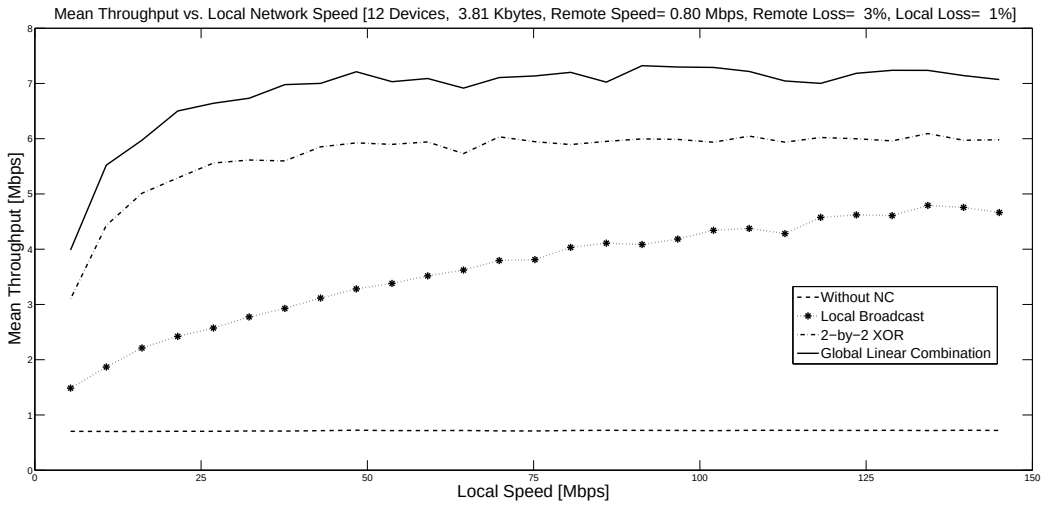


Figure 2.14: Mean Global System Throughput vs. LI Speed

Throughput vs. LI speed

In figure 2.14 the three algorithms are tested against the LI speed. The number of devices is fixed at 12 and we already know that 12 is the minimum number of devices for a 27 Mbps connection: the idea is to see how the network resources are used by the three algorithms.
The figure shows how the two NC enabled algorithms uses the available bandwidth much better the RLB algorithm, indeed RLB is the only that take advantage when the available bandwidth exceed the 27 Mbps value, offering in any case a worst mean throughput to the devices, with respect to the NC algorithms.

Real Implementation Parameters

As summarized into table 3.3, the real scenario is characterized by different values, with respect to the simulated ones (see table 2.2). The reason we does not use the same parameters in the two numerical simulation and physical scenario is that the simulator cannot handle in reasonable time the file that is used by the real devices.

As reference, we have decided to run the Matlab simulator with the real

parameters, with the following results grouped in the following table 2.3

Table 2.3: Numerical simulation with the parameters taken from the real scenario.

Simulated results					
	Internet	NC	Dup.	Corr.	Mean Speed
W/O NC	570	0	0	-	64 Kbps
RLB	392	178	1976	-	89 Kbps
2-by-2 XOR	213	357	1492	-	164 Kbps
GLC	209	361	1184	-	168 Kbps

Concluding Remarks

The RI speed impacts on every algorithms, limiting the amount of data available to every single device, as you can see in figure 2.10. Indeed in perfect case, without any packet loss, the amount of traffic served by the combination of the two interfaces does not exceed the 7.2 Mbps value: this is because of the RI speed of 0.8 Mbps is limiting the global system performance. Because of the "per device" nature of the RI max speed bound (S_r), in a situation with N devices, we can reach an upper bound for the information flow of:

$$S_t = N \cdot S_r. \tag{2.14}$$

In this case, this means an ideal speed of

$$S_t = 0.8 \cdot 12 = 9.6 \text{ Mbps} \tag{2.15}$$

The difference between the ideal throughput and the real one, can be explained by two factors. The first is the 3% loss probability on the remote network that can affect the performance globally: the less information going into the system, the less throughput can be reached by the system itself. The second factor is the casual selection of the remote chunks: maybe many devices are requesting the same chunk to the remote host, this way there is

redundancy in the chunks that flows into the global system. This means less information going into the system and, as in previous case, less throughput. Generally the second factor is more relevant than the first one in small sized files. In case of large files, on the contrary, the first factor becomes dominant.

In numerical simulated environment, the implementation complexity of the GLC is not balanced by a performance gain that could justify the implementation in a real scenario.

Chapter 3

Android Demonstrator

We have developed an application for commercial personal devices to show the power of the NC theory, called *Android Network Coding Demonstrator*. We have selected the Nexus 4 as the main development platform, even if the application is usable on every modern Android smart-phone, but also on any Android powered device, like tablets. In this chapter we illustrate the technologies used, we present the application and then we compare the results obtained with numerical simulations in chapter 2.

The main objective to be reached is to provide to a wider audience as possible a tool that shows the benefits of of the NC theory usage.

3.1 Technologies

To develop a fully functional demonstrator that can run on commercial devices, we have to find which are the technologies that can be used to obtain the best performance, in terms of data throughput, usability, and availability of this technologies by the end users.

In this section we present the platform on which the application will run and the main transmission technologies that have been used for the development.

Top Smartphone Operating Systems, Shipments, and Market Share, 2013 Q3 (Units in Millions)

Operating System	2Q13 Unit Shipments	2Q13 Market Share	2Q12 Unit Shipments	2Q12 Market Share	Year-over-Year Change
Android	187.4	79.3%	108	69.1%	73.5%
iOS	31.2	13.2%	26	16.6%	20.0%
Windows Phone	8.7	3.7%	4.9	3.1%	77.6%
BlackBerry OS	6.8	2.9%	7.7	4.9%	-11.7%
Linux	1.8	0.8%	2.8	1.8%	-35.7%
Symbian	0.5	0.2%	6.5	4.2%	-92.3%
Others	N/A	0.0%	0.3	0.2%	-100.0%
Total	**236.4**	**100.0%**	**156.2**	**100.0%**	**51.3%**

Figure 3.1: Android market share in August 2013, source: IDC Worldwide Mobile Phone Tracker, 7 August 2013

3.1.1 Why Android?

As platform we need an Operating System (OS), available for the majority of the devices, to let many people as possible to try out the NC demonstrator, with the more different device as possible, so we have chosen the Android platform.

Android is an OS based on the Linux kernel, and designed primarily for touchscreen mobile devices such as smart-phones and tablet computers. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance: a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. As shown in figure 3.1 up to now the 80% of the world shipment of mobile devices are Android powered.

Other key features that drives us to the Android platform are multiple:

- Android is fully open-source and provides free access to development tools and Application Programming Interface (API) documentation, allowing us to develop our demonstrator without any fee. It is very useful to have a complete API description available to be able to use

advanced device features like, the Wi-Fi interface, at their best. This is also useful to everyone they may want to improve or extend the functionalities of this demonstrator.

- Due to its open-source origin, all the aspects of the OS can be modified, can be improved for a specific demand, and is possible to know exactly how the OS manage the peripherals.

- Low budget development hardware: the Google reference board is the LG Nexus 5, priced at €350. As reference, Apple iPhone 5S costs at least €700.

- Android, like Apple's iOS and the most of the mobile OS, has a web-store to publish and download applications. Every Android device can access the *Play Store*, this is the name of the Android web-store, to find at any time new applications. Unlike other platforms, Android requires a single-time fee of 25$ to publish applications, the others requires an annual subscription.

So, why Android?
Simple, to reach as much people as possible. Our project has to be as most available as possible, so that anyone anywhere in the world can take benefit from the power of the NC theory.

3.1.2 Wi-Fi

Wi-Fi is a popular technology that allows electronic devices to exchange data wirelessly using radio waves. This technology is present on-board in every modern personal device. It is wide available, and this is the reason we have chosen this technology to act as LI

Wi-Fi networks are, though, characterized by the presence of a central device, called AP, that generates the wireless network and coordinates all the client devices attached. This might be a problem, because of the cloud-based non centralized infrastructure that we want to develop (defined in

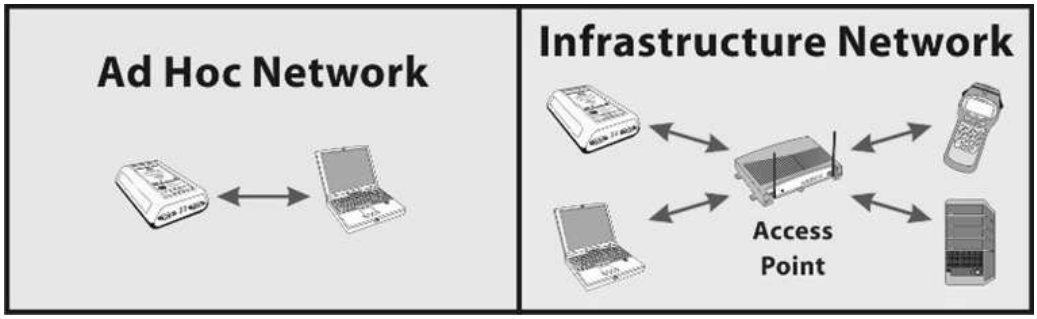


Figure 3.2: Wi-Fi Ad Hoc and Infrastructure mode.

section 1.1). Every Wi-Fi network, however, can support two configurations, namely *Infrastructure* and *Ad Hoc* mode.

Wi-Fi: Infrastucture vs. Ad Hoc

Conventional Wi-Fi networks are typically based on the presence of controller, the wireless AP. These devices normally combine three primary functions:

- Physical support for wireless and wired networking
- Bridging and routing between devices on the network
- Service provisioning to add and remove devices from the network.

This network type is called *Infrastructure network*. Transmissions in infrastructure mode are allowed only from and to the APs, none of the connected clients are authorized to transmit directly to another client. The AP, thereby, has an amount of buffer memory to temporarily store the received packet intended for other clients. The AP has to receive the whole packet in a time-slot, and then has to transmit to the destination, in another time-slot.

In case an infrastructure network is not available, the devices can self create a network, with features similar to the infrastructure mode. As shown in figure 3.2 the ad-hoc mode does not expect an AP. In ad-hoc network, each node participates in routing by forwarding data for other nodes, so the

determination of which nodes forward data is made dynamically on the basis of network connectivity. In addition to the classic routing, ad-hoc networks can use flooding for forwarding data. The presence of dynamic and adaptive routing protocols also enables ad-hoc networks to be formed quickly.

Wi-Fi ad-hoc mode is perfectly suitable for our purpose, but it is not officially supported by the Android platform. There are many unofficial ways to enable ad hoc Wi-Fi networking in Android releases, but because our goal is to have an as wide as possible public, we cannot relay on this feature.

Wi-Fi Direct

Wi-Fi Direct [9], initially called Wi-Fi P2P, is a Wi-Fi standard that enables devices to connect easily with each other without requiring a wireless AP and to communicate at typical Wi-Fi speeds and is fully supported in the 4.0 and up Android release, as for the Nexus 4 device that we use.

Wi-Fi Direct devices, formally known as *P2P Devices*, communicate by establishing *P2P Groups*, which are functionally equivalent to traditional Wi-Fi infrastructure networks [9]. The device implementing AP-like functionality in the P2P Group is referred to as the Peer to Peer Group Owner (P2P GO), and devices acting as clients are known as *P2P Clients*. Given that these roles are not static, when two P2P devices discover each other they negotiate their roles (P2P Client and P2P GO) to establish a P2P Group. Once the P2P Group is established, other P2P Clients can join the group as in a traditional Wi-Fi network. Wi-Fi Direct does not allow transferring the role of P2P GO within a P2P Group. In this way, if the P2P GO leaves the P2P Group then the group is torn down, and has to be re-established using some specific procedures. Into [26] you can find the precise definition of all the roles that a device has to implement to be Wi-Fi Direct compatible.

3.2 Network Coding Demonstrator App

In [8] is proposed an implementation of the network coding theory, based on Nokia Symbian devices. This demonstrator is dated 2008, when smartphones were not so cheap and so widely diffused. Nowadays Symbian smart-phones are extinguished, computational power is exponentially increased and technologies like Wi-Fi Direct are widely available. Other papers like [14], [17], [20], [23] and [18] also present mobile device based projects, implementing only to the original NC algorithm.

Draw inspiration from the demonstrator proposed in [8] and in [21], using a similar approach to implement the NC methods proposed in this thesis so as to prove the practical benefits of these new methods. Our proposal is even to exploit the fast global expansion of mobile devices to allow to as much people as possible to test NC advantages on their personal device.

3.2.1 Scenario

Our application is designed to accomplish a simple task: allow a group of in-proximity users to cooperate to grab a remote content on their personal devices. The idea is to use a system based on the idea presented in [24]. The scenario can be applied in many situations, like a conference room where participants have to get conference material from a remote host, or public areas in airports, train stations or stadium, where users can download data about the schedule of flight or trains, or the last replay of a goal in soccer game. The common factor, enabling cooperation, is the proximity, that is the only requirement for the system to work. In-fact newly formed cooperative cluster, namely the group of devices that cooperates, can offer each participating mobile device a better performance in terms of data rate, delay, robustness, security, and energy consumption in contrast to any stand alone device.

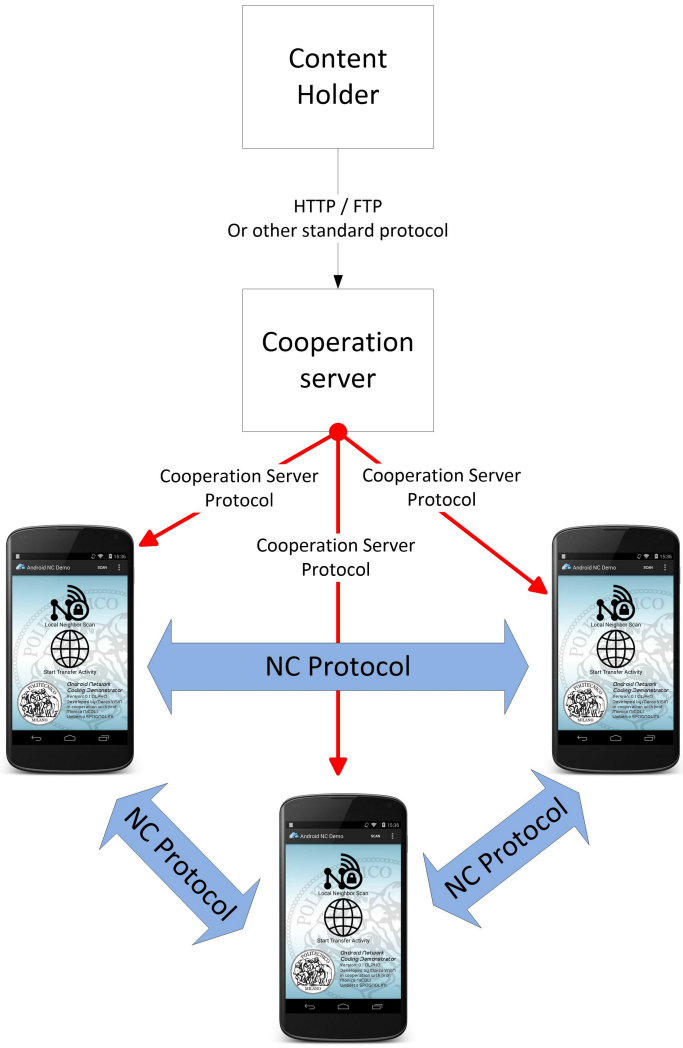


Figure 3.3: Cooperation server inserted into the general scenario.

3.2.2 Host side

A little overview of the server application is needed. In the model we suppose the host can segment the content in equal-sized chunks and can send one-by-one to the devices, on demand. The actual implementation of popular protocols, like HTTP and FTP, can be used to do this behavior, with some tricks: the idea is to use the *resume* function and to close the connection when enough data to build a chunk is received. This procedure has to be repeated for every chunk and is not really time efficient, because of the high overhead in opening and closing data connection both in HTTP and in FTP protocol.

This way a so called *Cooperation Server* is used, as shown in figure 3.3. This simple application run host side and only take care of segmenting the content and sending to the clients on demand in a secure and Cyclic redundancy check (CRC) protected way, the same way it was developed by Microsoft in [22]. Technically speaking, this is a simple shell application, written in C#, waiting for connection on a specific Transmission Control Protocol (TCP) port. The host side application only accepts direct unicast transmissions, like any standard Internet host.

3.2.3 Android side

As you can expect, the application is divided into two main segments, the same as the simulator:

- The Remote Interface (RI), used to get chunks directly from the cooperation server, though an unicast connection, using the Cooperation Server Protocol (CSP).
- The Local Intra-Cloud Interface (RI), used to share chunks across the local network that covers all the device involved, using the Network Coding Protocol (NCP).

The Android application can be summarized by the block scheme in figure 3.4, that indicates all the states that the application crosses during the execu-

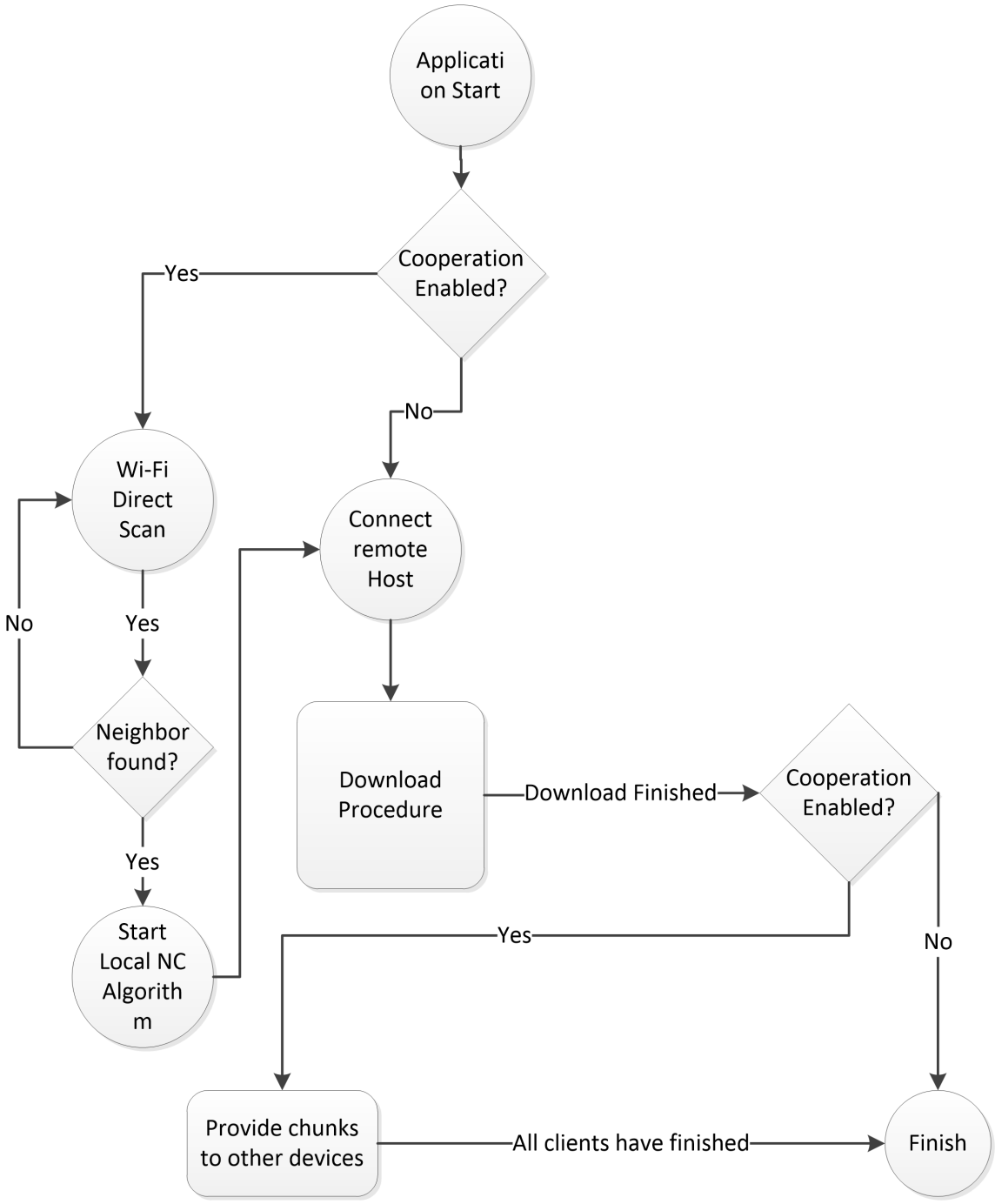


Figure 3.4: Block diagram of the Android application.

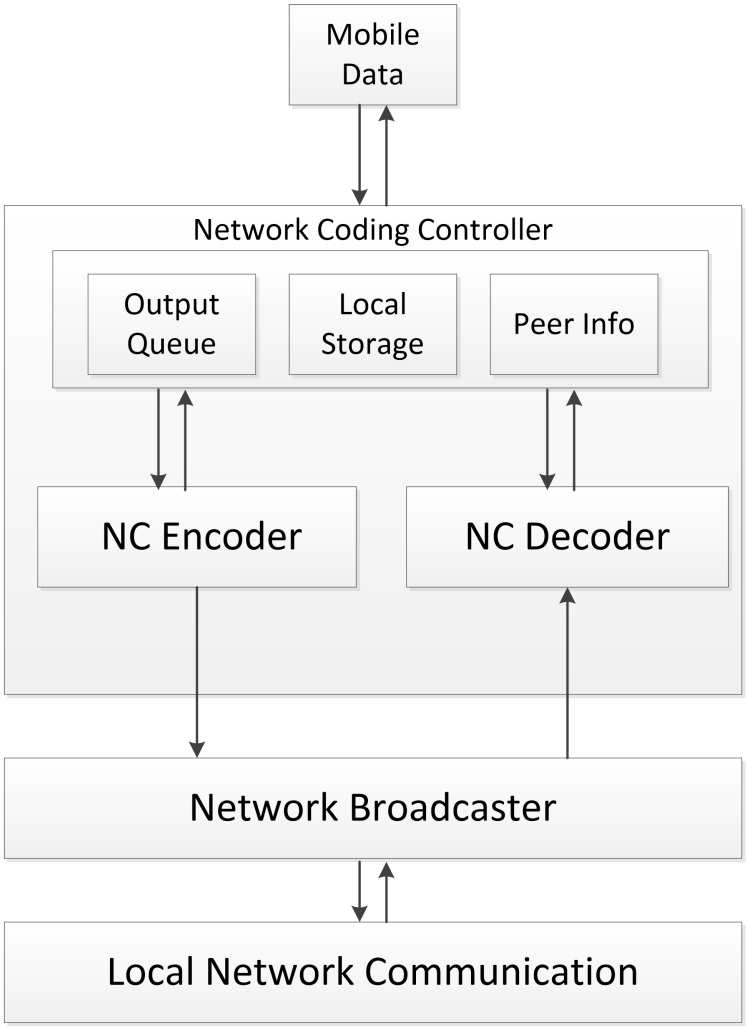


Figure 3.5: Schematic representation of the NC core engine.

tion. The download procedure itself is composed by many functional blocks, as shown in figure 3.5. As you can see, the *NC Controller* coordinates all the networking functions. The *Mobile Data* block is responsible for the communication to the remote host, while the Local Network Communication logical block represent the Wi-Fi card, in cooperation with the *Network Broadcaster*, that is simply the broadcast socket. The most important things are in the NC-Controller, where take place the real NC encorder and the decoder, which implements the chosen NC algorithm. To support the couple of encoder and

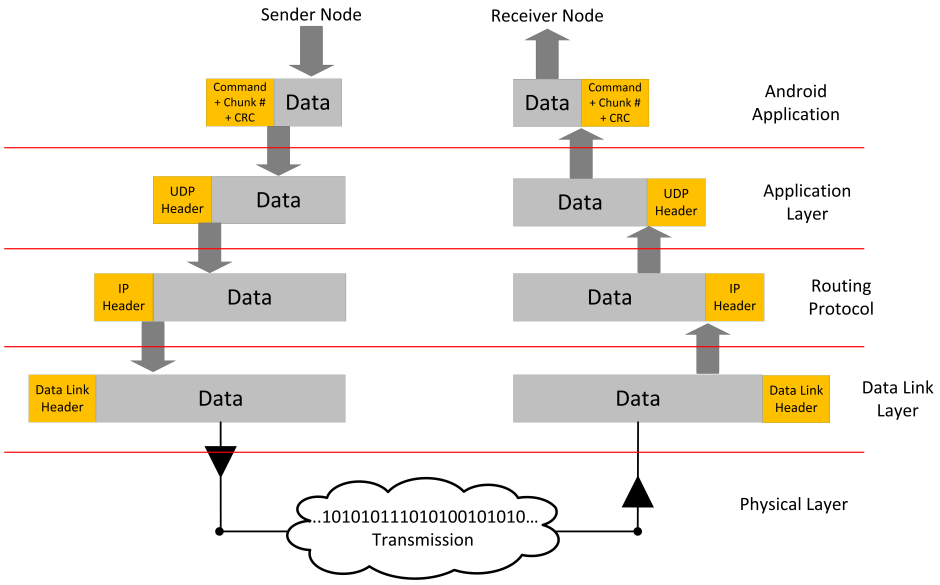


Figure 3.6: Network Coding Protocol (NCP)

decoder, there is the *Local Storage* block, which allow the application to save the received data locally. The *Output Queue* block is a FIFO buffer allowing the system to queue all the packages to be transmitted in the local network until a valid time-slot is found. Finally the *Peer Info* block is responsible to maintain updated information about all the clients composing the local distribution network.

3.2.4 Network Coding Protocol (NCP)

To develop this application a new protocol has been created over the top of the UDP/IP stack. Therefor the only level defined, in the ISO/OSI stack, is level 5.

This protocol is used by every device in the local network, to exchange data with the others. The definition of such a protocol allows to create compatible applications for different device type and/or OS. As shown in figure 3.6, physical, data link, routing and application layers are still the standard for any User Datagram Protocol (UDP) over IP packet, this means that every NC coded packet can flow freely in any existing routing network. Because of the

Wi-Fi data link layer, MAC, is an evolution of the Ethernet MAC, it inherits the maximum payload size of 1500 bytes. The routing layer (IP) introduces an additional 20 to 60 bytes long header and the application layer, UDP, adds another 8 bytes long header. Thus the max allowed payload length for an UDP packet is $1500 - 60 - 8 = 1430$ bytes. Our protocol generates a 1400 bytes long packet, thus it will perfectly fit the available space in the UDP payload. The NCP packet structure is defined as in figure 3.7. The first

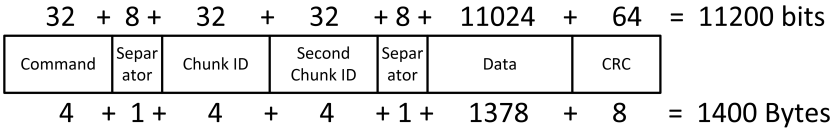


Figure 3.7: The packet structure of the NCP

field of the packet is the *Command ID*, a 32 bit long integer, indicating the purpose of the packet. The general scheme is to use $100 - 999$ commands, divided as in table 3.1. A separation character is used to detect if the packet

Table 3.1: NC Protocol commands.

Commands			
	Query		Response
100	Device Name Request	150	S/N(String), Phone Name(String)
110	Connection Info Request	160	Connection Info (String)
200	File Transfer init. Sequence	200	ACK (null)
201	File Size Request	251	File Size (long)
202	File Name Request	252	File Name (String)
249	Send Terminated		
253	Missing Chunk Signalling		
299	Reception Complete		
300	Packet Request		
500	Data Packet		

is correctly formatted: is necessary because the command header is not CRC protected. We need a *Chunk Indicator* field, 8 bytes long, to let the other devices know which is the chunk contained in the packet. In case the packet contains the XOR version of two different chunks,a second 8 bytes long field is available to specify the second Chunk ID. In the Android implementation of the NCP, on the contrary of what happens into the numerical simulator, chunks are indexed and only chunk index are transmitted, not the full **g** vector as explained in section 1.4.3. The reason is that only GLC algorithm really needs the entire **g** vector to be transmitted and because it is not implemented in the Android demonstrator, it is useless to transmit such a long information, chunk ID is enough. Another separator indicates the start position of the data segment, containing a chunk. The data segment is 1347 bytes long and is protected against corruption by a CRC, which occupies the last 8 bytes of the packet. The only command that requires a special packet format is the *Missing Chunk Signalling* , because its payload consists in an array of 32 bit integers, indicating the indexes of the blocks that are missing to the sender. This way any device in the network can have a global overview of the chunk distribution across the whole network. This feature is exploited by the 2-by-2 XOR algorithm (see chapter 1.4.2) to maximize the throughput.

3.2.5 Cooperation Server Protocol

As shown in 3.2.2, every device in the cooperation network, if is enabled to perform remote connections, can retrieve chunks from the so called *Cooperation Server*. To talk with the cooperation server, the Android application use another protocol, called *cooperation server protocol*.

In figure 3.8 is represented the protocol stack, that is the same as for the Network Coding Protocol shown in 3.2.4, but we are now using a TCP connection, instead of the UDP one.

This difference is due to the unicast nature of this transmission. Because this is a single source and single destination transmission, we can use TCP,

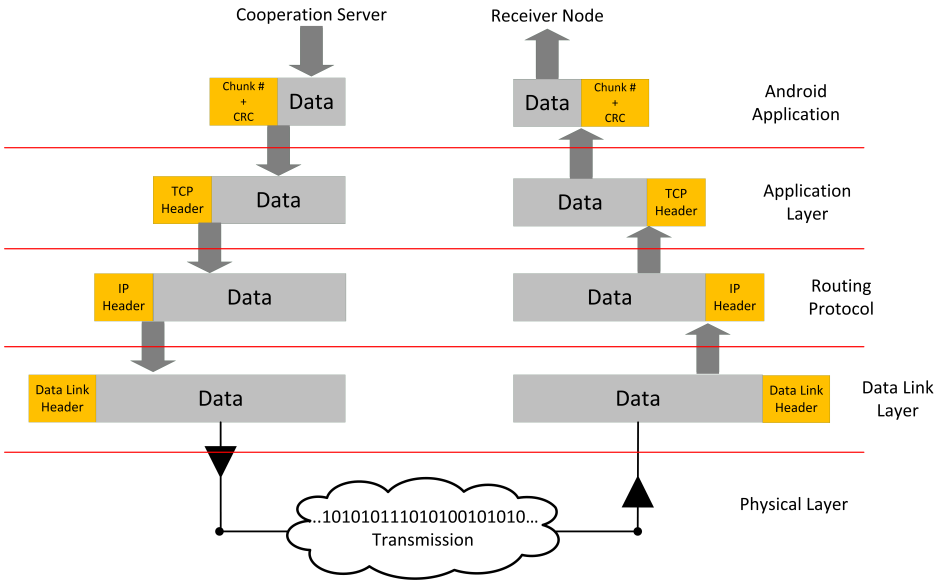


Figure 3.8: Cooperation Server Protocol

which is, in contrast with UDP, a connection oriented protocol. This means there are some control packets dedicated to open and to close the connection. This initial overhead does not influence the throughput of the connection, because it is an one-time fee to pay, but the TCP header is much larger than the UDP one: 20 to 60 bytes for TCP when only 8 bytes are required for UDP. Typically a TCP header is 20 bytes long, that is 16 bytes longer the UDP one.

Despite this additional overhead, the chunk size has been calculated to allow the cooperation server protocol to carry one chunk in a single packet: the summation of all the length does not exceed the maximum size allowed by the Wi-Fi MAC.

The TCP connection, however, guarantees the correct reception of packets by the clients and this feature is much more important than the overhead needed by this type of connection.

The packet format is changed too, as shown in figure 3.9, because the optional field indicating the second chunk present in the packet is gone. This means, a 4 Bytes lighter packet. In table 3.2 is reported the commands

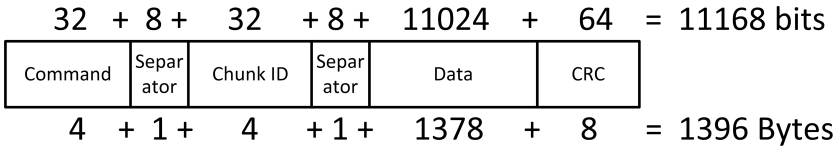


Figure 3.9: The packet structure. of the Cooperation Server Protocol.

scheme used by the cooperation server protocol. It is a light version of the network coding protocol (table 3.1), where are removed client specific commands, useless in this case.

Table 3.2: Cooperation Server Protocol commands.

Commands			
	Query		Response
201	File Size Request	251	File Size (long)
202	File Name Request	252	File Name (String)
249	Send Terminated		
299	Reception Complete		
300	Packet Request	500	Data Packet

Why CRC field in a TCP payload?
The response is a little bit strange and is widely discussed in section 3.4.3. By the way, the problem is related to the way in which it is used the Wi-Fi equipment.

3.3 Running on Real Device

After the technical introduction, we can present the application, running on a real smart-phone, an LG Nexus 4, but the application runs fine on every Android 4.0+ device.

The main aspect we focused on during the application development is the

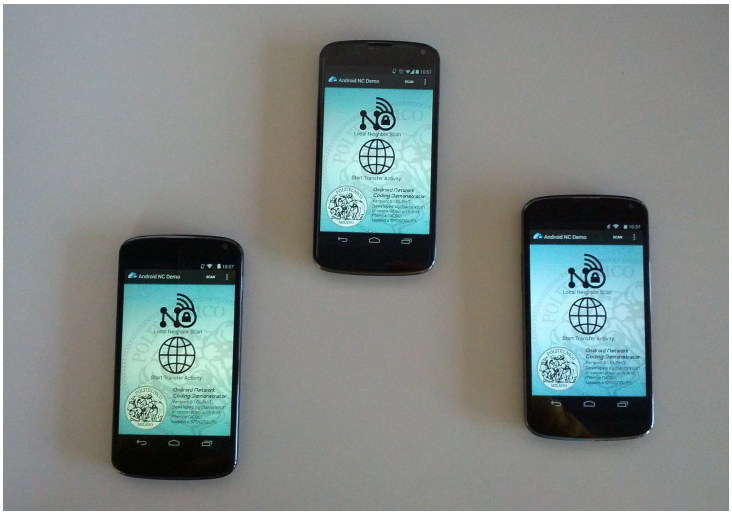


Figure 3.10: The real test environment, with the three Nexus 4 phones ready to transfer.

simplicity. The user has to be guided through the required steps starting from the procedure to establish the connections up to the downloading phase. The user is supposed to be completely unskilled about both Android phone and the NC theory, the application has to be as much user friendly as possible.

The application requires the presence of a Wi-Fi Direct enabled network card, generally present on every smart-phone equipped Android with version 4 or better. To connect to the remote cooperation server, an Internet connection is needed. In our experimental case we used a Wi-Fi network to provide Internet connectivity to the devices. The use of a Wi-Fi network even to provide connectivity involve some issues, as explained in section 3.4.3.

3.3.1 Application Life-Cycle

At the application startup, as in figure 3.11, the main screen is presented, where the user can choose one of the two main functionalities: search for neighbors or start directly the download procedure. Obviously in case the user choose to start the transfer immediately, he will not use any NC feature.

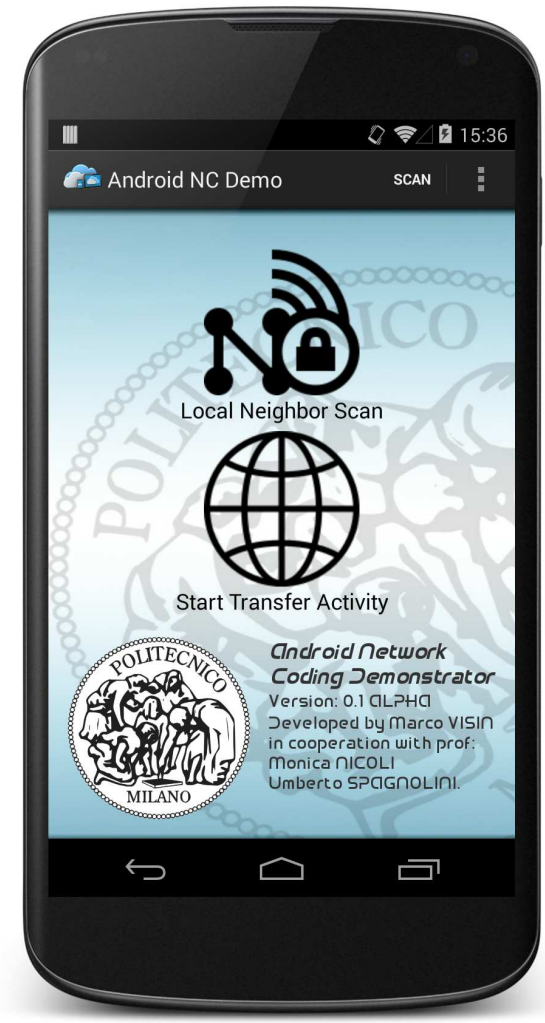


Figure 3.11: Android Application, main screen

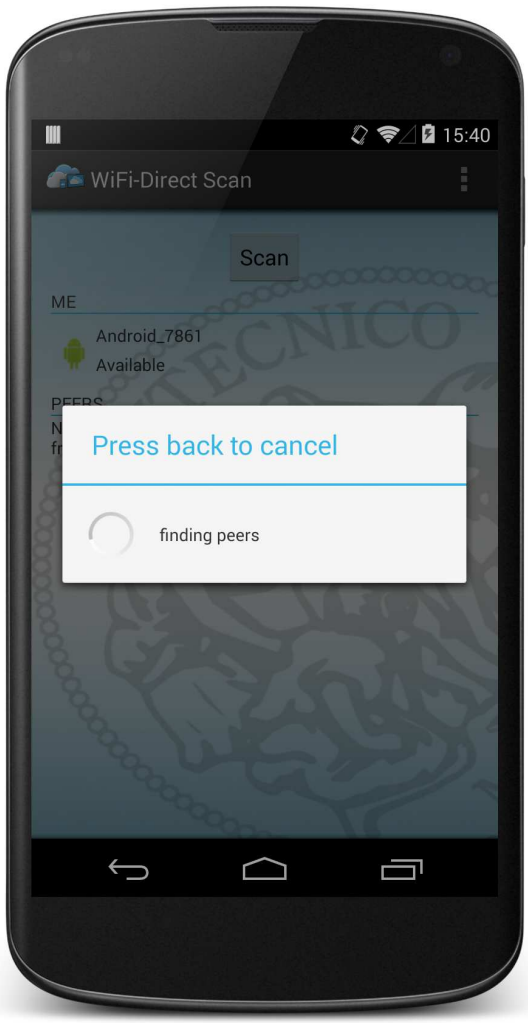


Figure 3.12: Android Application, Wi-Fi Direct scan screen

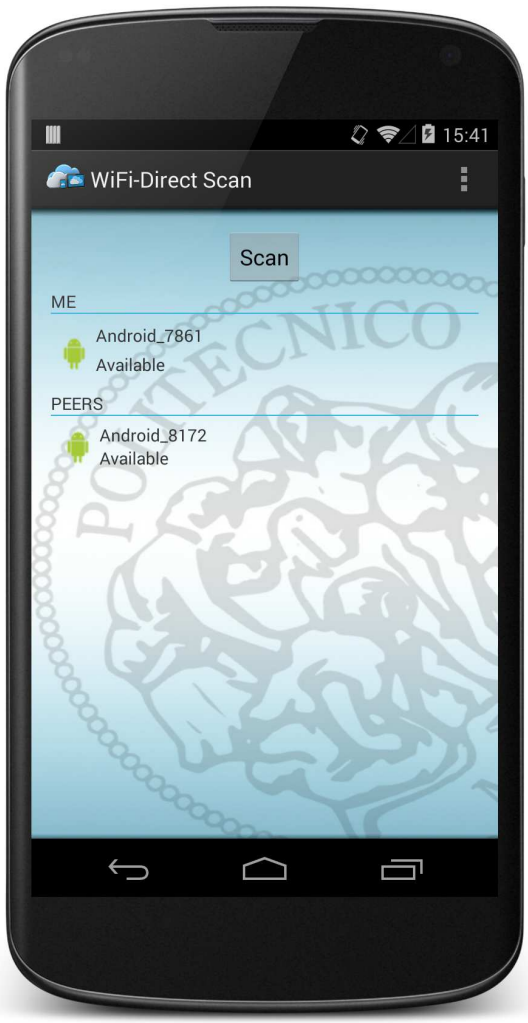


Figure 3.13: Android Application, Wi-Fi Direct peer result screen

The local neighbor scan, shown in figure 3.12 and 3.13, gives the user the opportunity to select another device in proximity, to start a shared download. As explained later in section 4, the selection of the local peers can be automatized. After the peer selection, the application will show directly the transfer screen, because this is the only required procedure, before the download can take place.

When connecting to a P2P GO, the application shows information about the P2P Group, such as the P2P GO IP address, the number of the group participants and the IP subnet associated to the P2P Group. This information is useless for normal users, but it is fundamental to a developer that wants develop a compatible application for other platforms. With such an information, the developer can check the correctness of the group join procedure he is developing This is the reason why they are showed during the pairing procedure.

In figure 3.14 you can see what is the output from the application: the user is informed about the global number of chunks to be transferred (depending on the file size), the number of chunks received directly from the remote host through Internet, the number of chunks received by NC technology, in the local broadcasting network, the number of duplicated chunks received and the number of corrupted chunks. In-fact maybe in some chunks CRC check fails as they are not correctly received. That packets are discarded and counted in this row. The user is also informed about the actual speed of the information flowing into the personal device, regardless of the method used. The application allow the user to measure in a precise manner how much remote traffic that has been generated by the transfer and at the same time, how much traffic has been spared.

One of the most interesting feature of this view is the real-time data visualization: the user can indeed see in real time how much data is coming from every source type, as shown in figure 3.15. As you can see, in the specific case, we are transferring the lena image with cloud network composed by two devices. The performance analysis are done in the following section, but you

Figure 3.14: Android Application, transfer screen

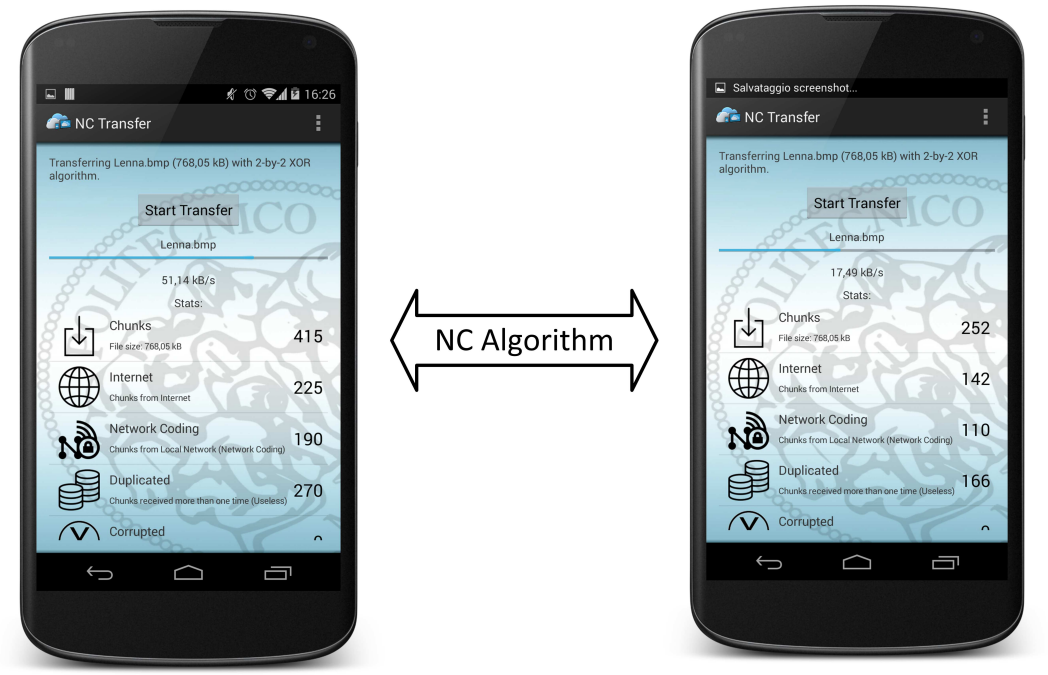
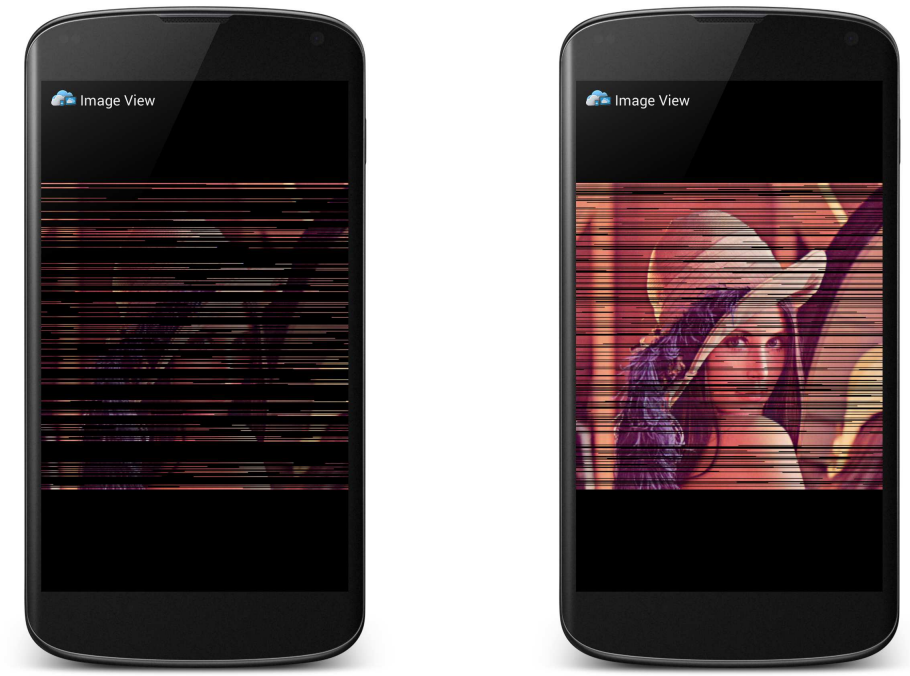


Figure 3.15: Android Application during a transfer with the 2-by-2 XOR algorithm and a cloud network composed by only two Nexus 4 devices

can see even in such a small scenario, the number of the chunks obtained from the remote interface is half than the total received. The figure also shows that the actual speed can be much higher than the RI one which is in this case 64 kbps = 8 kB/s

The application is designed to download the content into the mass memory of the phone, allowing the user to inspect the downloaded data with an external application to check the correctness of the data. In case the data is an image, is possible to open an image view, showing visually the download progress as shown in figure 3.16.



(a) Device without NC enabled (b) Device with NC enabled

Figure 3.16: Android Application, image view screen during the "Lenna" image transfer.

The images are captured after the same time from the start of the transfer.
The cloud network is composed by 4 "Nexus 4" devices, using the 2-by-2 XOR NC algorithm.

3.4 Results Analysis

We expect to see an improvement on the mean throughput for every NC enabled device, as much as a drastic reduction in the number of packets obtained from the RI. We expect, for this reason, a minor download time, with respect to the single non cooperative download.

3.4.1 The Real Case Scenario

The scenario in which the application is tested is the same that we have simulated and is explained in section 2.1. In this case we put some smartphones side by side, connected to the same 802.11 G Wi-Fi network, acting as RI. At the same Wi-Fi network we attach the cooperation server which holds the content and prepare chunks. Then the devices will create another Wi-Fi network, using Wi-Fi direct, to be used as LI. At this point the phones are ready to start the download procedure, using all the same algorithm at a time.

As already exposed in section 2.3.4, the real scenario is different from the one we have used in numerical simulations. The reason is that to show a real advantage in NC usage in the Android application, we need to use a larger file, with respect to the one we have used into the numerical simulations. Another difference between the two numerical simulated and real case scenario is that due to the problem explained into section 3.4.3, the Cooperation Server has to impose a transfer rate of few kbps. So the real Internet connection provided to the devices is very slow, comparable to a GPRS one.

The application results are tested against the numerical simulated ones, to check the correctness of the implementation. During comparison, we need to remember the numerical simulation limitations, regarding the scenario and the implementation itself, exposed in 2.2.6.

The parameters we have used in are summarized in the following table 3.3, to simplify the comparison with the numerical simulation.

Table 3.3: Parameters of the real environment.

Parameters	
File Size	768.05 KBytes
Number of "Nexus 4" Devices	4
Number of Runs	10
Chunk Size	1378 Bytes
RI Speed	64 Kbps
RI Loss Probability	Unknown
LI Speed	27 Mbps
LI Loss Probability	Unknown

3.4.2 Results

The download procedure is executed 10 times for every algorithm, results are averaged as in the numerical simulations. The results are grouped in table 3.4. You can see the number of chunks downloaded by the RI (Internet), by the LI (NC), the number of duplicated- (Dup.) and of the corrupted chunks (Corr.). The last column represents the mean throughput, i.e. the mean of the transfer speed measured by the devices. The duplicated chunks are calculated as the number of transmissions that are correctly received by the node, but does not add any information to the node. For example the node already owns the chunk contained into the specific transmission. The corrupted chunks are calculated as the number of transmission in which the CRC integrity check fails. In the table are grouped the results from the direct linear download from the remote host (W/O NC), with the reference RLB algorithm and with the proposed 2-by-2 XOR algorithm.

As you can see in table 3.4, the results are comparable to the numerical simulated (see table 2.3). This means it is really possible to implement the 2-by-2 XOR algorithm into personal device with performance improvement for the user. There is a 14 % difference between the theoretical simulated and the practical implementation, this can be explained both with the aspects

Table 3.4: Android application results, in real case scenario. Is summarized the number of packets obtained by every interface, with the number of duplicated- (Dup.) and corrupted packets.

Application results					
	Internet	NC	Dup.	Corr.	Mean Speed
W/O NC	570	0	1	0	63 Kbps
RLB	387	183	1786	254	90 Kbps
2-by-2 XOR	250	320	1552	124	140 Kbps

illustrated into section 3.4.3 , either with the limitation introduced by the simulated scenario (see section 2.2.6). In a nutshell the difference is that the signalling traffic is not present into the numerical simulator, thus the real performance are poorest.

Regarding the performance of the other two modes, the direct non cooperative mode reaches 63 Kbps, proofing that the cooperative server successfully limits the bandwidth, and surprisingly, the RLB algorithm marks an identical (or slightly better) mean speed than the numerical simulated version. The reason for this type of result is that the random choice of the chunk to be transmitted generates a so poor throughput that the loss in performance of the real implementation explained into the following section 3.4.3 is not noticeable.

Generally speaking, the improved data rate, delay, and robustness comes obviously by the global accumulated remote links with its inherent diversity. Regarding the energy consumption [16], in a nutshell, as long as the energy per bit ratio is better in the short range connection than the cellular link, the cooperation will help in sparing energy on the devices. In fact the number of chunks (and consequently the number of bits) transferred by the RI is less in a NC enabled environment than in a typical one. Since the path loss is much smaller on the short range communication (around 10 m) typical of local environments than the cellular one (several 100 m), with the NC adoption

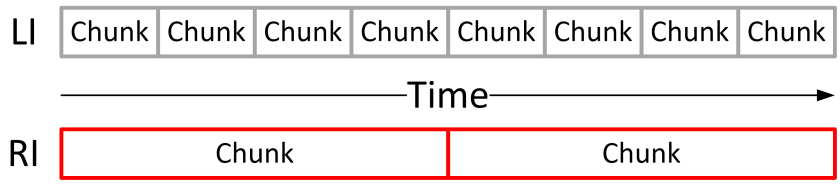


Figure 3.17: The classic approach with two network adapter.

we will spare energy regardless of the local wireless transmission technique used, in present as much as in future technologies. For cooperative wireless networks the proposed 2-by-2 XOR NC algorithm seems a viable solution to decrease the traffic within the short range cluster and with that to decrease the delay and the energy consumed.

3.4.3 Real-Case Issues

As told before, the real Android implementation suffers of an issue. Normally the application is used assigning the two RI and LI to different physical network adapter on the device, as in figure 3.17. As the typical configuration, the RI is the cellular data connection, while the LI is the Wi-Fi network adapter. The problem is that we can use the wireless network into two different mode at the same time. The local Wi-Fi interface is used to generate, or join, a Wi-Fi Direct P2P Group to exchange chunks with the neighbors, and this is the key point of the application itself. The problem is that, as in our scenario, the device can also be at the same time connected to another Wi-Fi network (in infrastructure mode), to get Internet connectivity. This is possible by setting the Network Interface Controller (NIC) to switch at a fixed time rate, from the channel used by the P2P Group to the one used by the Wi-Fi *infrastructured* network and vice versa, like in figure 3.18. Even if the Android API documentation does not report any issue about this configuration, our application reveals that the NIC suffers of some errors.

As result, the NIC switches the channel configuration even during the reception of a packet. This results in packet corruption. This is the reason of the high number of corrupted packet signaled by the application, because

Figure 3.18: In case the RI and the LI are assigned to the same network adapter, the Network Interface Controller has to switch between the two Wi-Fi channels, when it is required.

all the UDP local messages are not a priori integrity-checked and thus the application build-in CRC check fails. This is an issue that degrades the throughput achievable by our application.

Into the RI side, however, this issue creates a very strange situation. Indeed, the RI relays on a secure TCP connection through the Internet, to the cooperation server. One of the most important and characteristic feature of the TCP stack is to check for the integrity and for the sequentiality of the received packets. This means that the packets are numerated and there is a re-transmission system that can be used to retrieve lost packets. In figure 3.19 is shown an example of a broken transmission. In this specific case, the transmission of the packet number four fails. Since TCP packets are numerated, the receiver notices the lack of the fourth packet and asks for the retransmission. This process is completely transparent to the upper ISO/OSI application layer, thus the application relaying on a TCP connection receives always all the transmitted packets, in the right order. In case the Android NIC is forced to work over two different channels, like in our case, this mechanism manifests some issues. Indeed the TCP transmission acts exactly as an UDP one: there is no sequentiality control and, most important, there is no transmission error control. As a consequence we found out of sequence packets and corrupted packets at the application layer; in case of out-of-sequence packets, there is no problem, the algorithm does not require a specific order in transmissions, but it is really important to detect broken packets. This is the reason why, even into the cooperation server protocol, there is a CRC field to check integrity client side and discard broken packets. Without this check, indeed, invalid chunks propagates across the

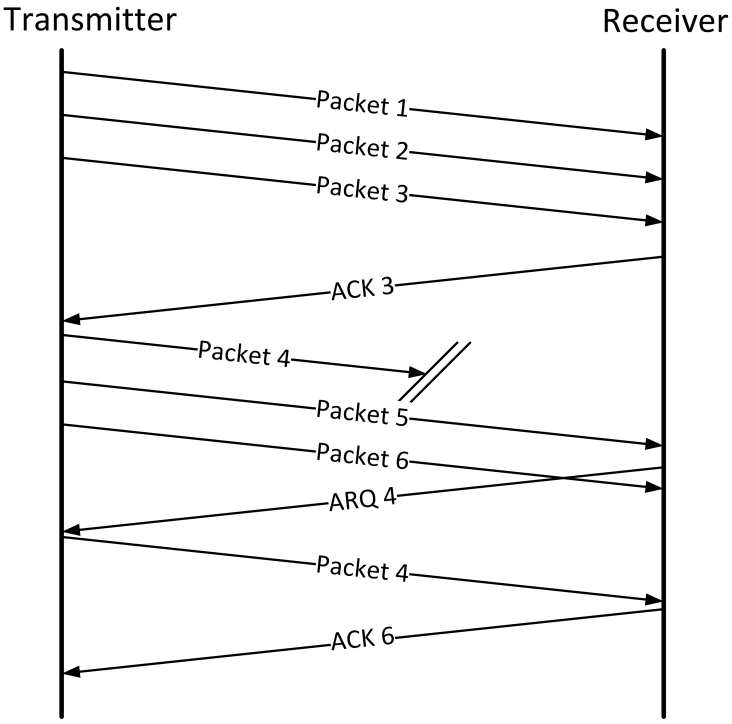


Figure 3.19: TCP packet transmission sequence with transmission error. When the receiver receives an out of sequence packet, it requests the retransmission of the lost one.

In this example we suppose that the acknowledge (ACK) is sent every three transmission; the ACK packet confirms, as usual, the correct reception of all the packets until the number it carries.

Selective Repeat is one of the automatic repeat-request (ARQ) techniques; with this technique the receiver asks for the retransmission of a single specific packet.

whole cloud network, compromising the decoded content on all the devices.

This issues can be simply avoided using a cellular network connection to provide Internet connectivity to the device. Unfortunately this is not possible at this moment as it would be necessary one SIM for each device. The way we use to avoid this issue is to limit bandwidth into the cooperation server, to let the NIC of the devices to stay as much time as possible into the P2P Group channel, loosing the smallest number of packet that is possible. A good compromise is to limit the cooperation server bandwidth to 64Kbps, as in our case, simulating a GPRS cellular link. More bandwidth will result in non coherent results: depending on the random time in which the NIC switches channel, the application gives out the expected number of packets obtained from NC, as well as a small number, which may also tend to zero.

Chapter 4

Conclusions and Future Work.

This thesis has been realized with the aim to go beyond the limits characterizing the actual available mobile communication systems. Starting from the NC theory present up to now in literature, we have proposed an alternative approach, based on the XOR operation. Accordingly, we have characterized the scenario of application. Then we have summarized the actual NC theory to highlight the key points that we can use into our original algorithm. The focus is then moved to the 2-by-2 XOR algorithm: it has been explained which are the benefits in using the proposed algorithm and which are the key differences with respect to the classic NC theory. The algorithms are then numerically simulated into a specific scenario, to justify the implementation on real devices. We observed that the numerically simulated results are coherent with what we expect, as a consequence the proposed algorithm is deployed into real devices to test the performance in a real scenario. An Android based application has been developed with this aim. Finally, the results collected from such an application are compared with the ones obtained from numerical simulations, to validate the algorithms in a real scenario.

The analysis work conduct in this thesis has demonstrated how the proposed NC algorithm, 2-by-2 XOR, is suitable to be deployed into real devices for everyday usage. It offers more than the double of the throughput with respect to the direct-, single device download. It is more than 1.5 times faster

than the reference RLB approach. The advantage with respect to the GLC algorithm, taken directly from the NC theory, is evident both in terms of implementation complexity, and in the computational power requirements. Although is requiring some additional signaling traffic, the proposed 2-by-2 XOR algorithm does not require any complex computation at the device. Thus the downloaded content is immediately available to the device, when the download is completed. This also compensates the minor throughput offered with respect to the GLC algorithm.

Future Work

Here are some possible future field of application for the NC theory and in particular for the proposed 2-by-2 XOR algorithm.

Automatic Peer Selection

The first idea to improve the proposed application is to implement the automatic selection of nearby peers. As a consequence, the user has not to select manually a nearby device to enter the cloud network, the system will do it. This is the first step to create an OS based NC helper, that can provide NC benefits on each transmission done by the device.

Pervasive Mobile Device

A possible future development direction for the proposed 2-by-2 XOR algorithm is the adoption in many mobile device OS, like Android, Windows phone and similar. This will guarantee an improvement over the current network performance in downloading common contents, like OS and application updates, generating at the same time less traffic to the server network. Implementation into the device OS can allow all the applications to use the NC technology in a transparent manner.

Motes

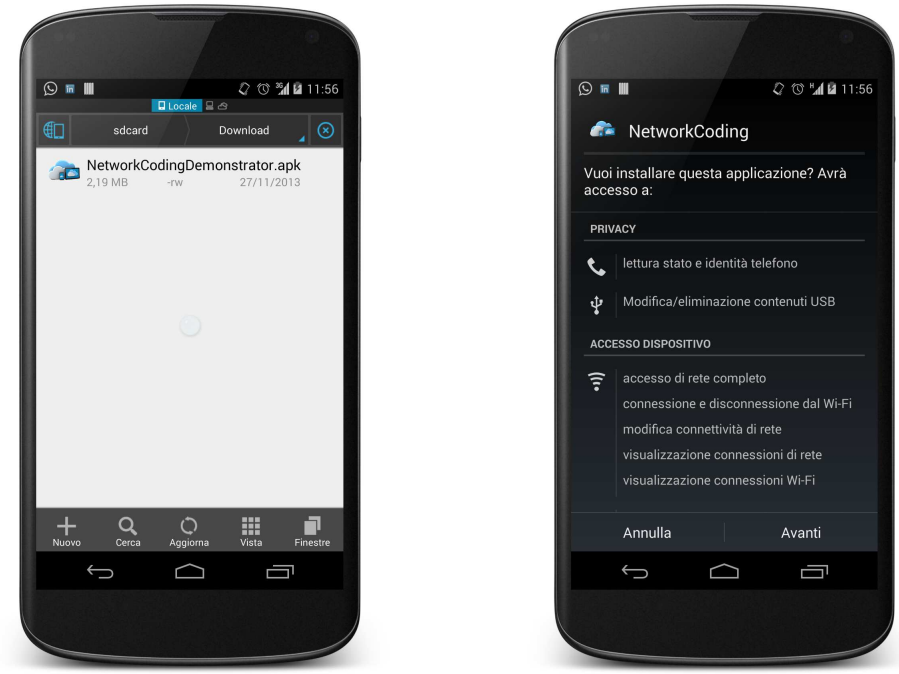
By definition a mote is a sensor node in a wireless sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network. Motes are characterized by small size, poor computational power and low power consumption. The 2-by-2 XOR approach is usable even on motes, in contrast with other NC algorithms, because only requires that the mote can compute XOR operations, which is typically a low power-demanding operation, often it is hardware implemented. With this technique it is possible to give the new generation sensor network the possibility to preserve energy, maintaining active the radio interface less time.

Automotive

Another field of application for the proposed algorithm is the automotive segment. In the future car-to-car communication system will be adopted by every vehicle available in the market. They will be useful to share information about the traffic situation as well as the road condition and many other useful information for both the passenger and the car itself. An example could be the satellite navigator that automatically adjusts the route to avoid queues. Another example can be the automatic balance adjustment; the car can share information about the road to allow the following cars to automatically set some parameters into electronic controlled suspensions to let the passengers to have the maximum comfort. All this information exchange can benefit from the NC technique usage. In fact, the 2-by-2 XOR algorithm improves the robustness of the transmissions in an environment with high loss probability. As a consequence, the car-to-car communication is made possible even if the communication links between cars are not reliable. The less network latency in an 2-by-2 XOR enabled environment can be useful to a faster propagation of the information across the cars, allowing a well-timed change in route.

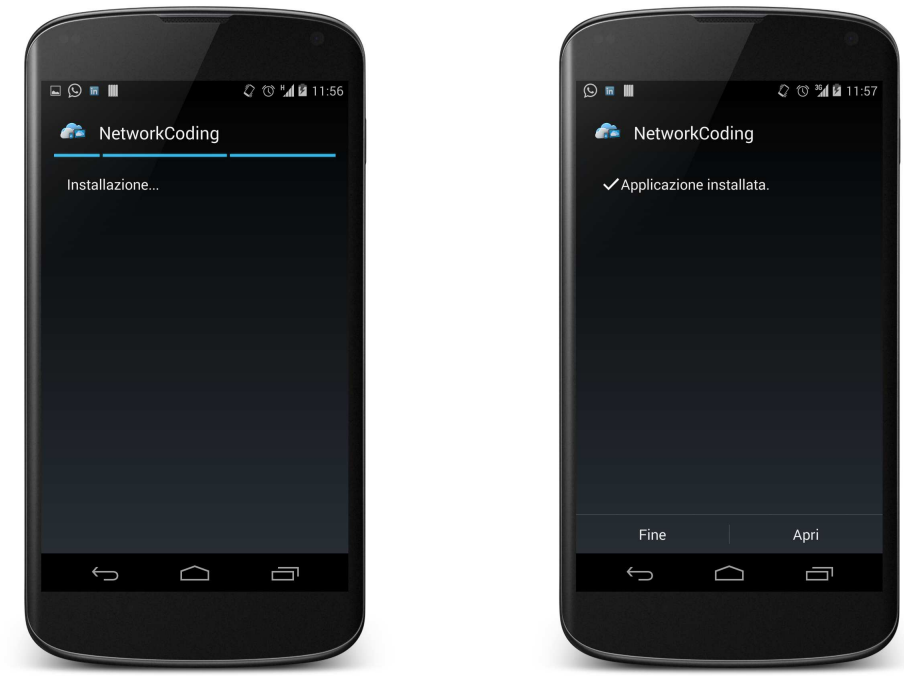
Appendix A

In this chapter we analyze the *Android Network Coding Demonstrator* application installation and usage.



(a) File selection (b) Permissions request

Figure 1: Android Application installing procedure (Part 1)



(a) Application installing (b) Installation finished

Figure 2: Android Application installing procedure (Part 2)

The installation of the application is simple as any other installation on an Android based device but until the application is published to the Android Play Store, you have to manually download the application as an APK file. In figures 1 and 2 you can see the installation process.

The installer remember us that the application require the permission to use some features of the device. In table 1 is a short review of the permissions required with the reason why we need that specific permission on our application.

When the installation is complete, you can start the application as conventional applications. In figure 3 is illustrate the main screen. The Start Transfer Activity button skips the NC configuration and directly starts the download procedure with an unicast connection. Alternatively, if you chose

(a) Main Screen

(b) Peer Selection

(c) Application installing

Figure 3: Android Application Life Cycle

Table 1: Application permissions requirements.

Permissions required by the application	
Permission	
Full network access	Required
Connecting and Disconnecting Wi-Fi networks	Required
Network Connectivity	Required
Check Network Connection	Required
Check Wi-Fi Connection	Required
Read Phone Identity	Required
Read/Write External Storage	Required

to scan for local neighbors, you have to select one peer from the list of the available neighbors. You will be connected to that peer specified and the two devices automatically negotiates the roles they will have in the P2P network. In case the selected peer already belongs to a P2P network, you will be added to such a network to share chunks with all the pre-existing nodes.

The transfer view presents the information the application outputs during the download. In this view is indicated the total number of received chunks and the division between locally obtained - by performing NC cooperation techniques - and directly received chunks, from the cooperation server. Other available information is the number of duplicated- and corrupted transmissions received, as well as the total mean incoming speed.

In figure 4 is possible to see the options view, that is created to allow the user to set some important parameters for the application execution. It is possible to enable or disable the cooperation procedure. When the cooperation procedure is enabled, it is required to select the algorithm to be used. It is also required to insert the address of the cooperation server for the remote interface. There is also an option to keep the screen active during the transfer, to let the user to check the download progress without having to unlock the phone.

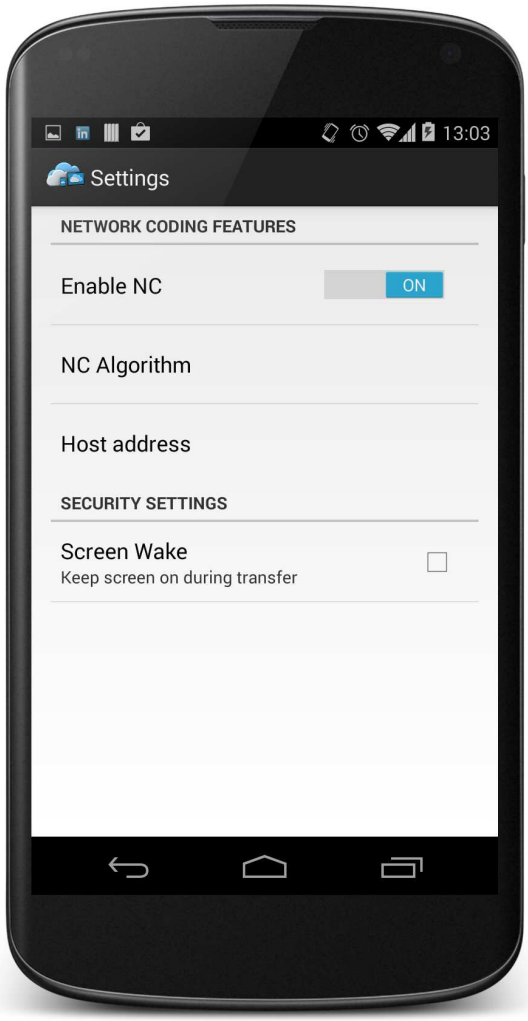


Figure 4: Application Options View. Here you can specify if you want to use NC and in case which algorithm you want to use. You have to specify the address of the Cooperation Server

In figure 5 you can see the last screen available during the download: the image view. In case the selected file to be transfered is an image, you can select this view from the options menu and you can visually observe the download progress. The image is composed while the chunks are correctly downloaded by the device. When the transfer is complete, the downloaded file is saved into the mass storage of the device, to let the user to check the correctness of the transfer.

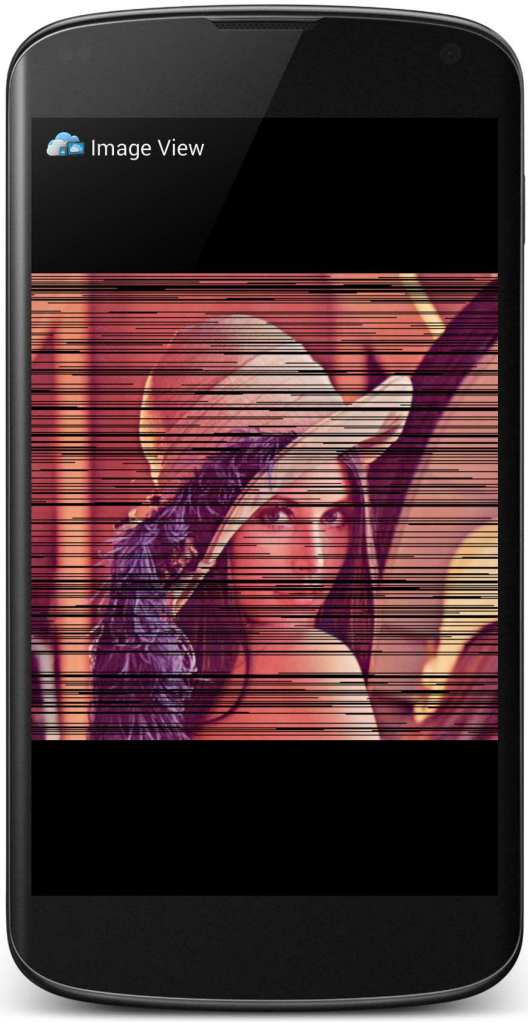


Figure 5: Visual transfer progression. The image is composed while the chunks are downloaded.

Appendix B

In this section we illustrate the code analysis for the Android demonstrator. The application is written in Java-code, as the only programming language supported by the Android platform. The User Interface (UI) parts, not analyzed in this quick review, are written as traditional XML code.

The Android application code is functionally divided in many class as this is useful to logically divide the application structure into functional blocks, that are working together. This type of structure is widespread because it helps into errors preventing and correction, as it is possible to test all the classes individually.

The key point of the presented structure is the *NC* class, which is an abstract one. This means that in this class is only defined the structure of the class itself. The practical implementation has to be done in a separate class. As a consequence the developer can add any type of coding scheme, extending this abstract NC class with proprietary code. The NC abstract class defines also some functions that are common to all the NC algorithms.

The main classes of the project are analyzed in the following pages, presenting also the relative UML diagram.

Device class

The first class we analyze is the Device class (Figure 6), which represent a device in the cloud network. The class contains all the useful information about the specific device, like a name and S/N field, useful for device identification. It is also present a field for the IP address assigned to the device, this is used to identify the received transmissions. The Device class supplies functions to set chunks that it owns, this means that when the application receives a service transmission by this specific client which contains the list of the devices' missing chunks, this function will be used to update the chunk status. The most important function that this class provides is the *getMissing()* that returns the list of missing chunk for the device.

The *Devices* class manage the list of devices representing all the nodes that belong the cloud network. The functions are an extension of the previous ones and require the address of the specific device to query. The *Timer* defined is used to actively ask for chunk status update to the neighbors.

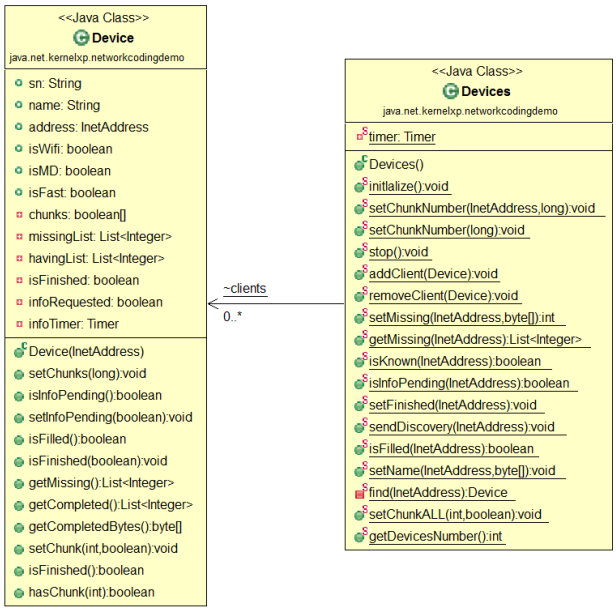


Figure 6: Device class scheme

FileSave

Figure 7: FileSave class scheme

The *FileSave* class (Figure 7) is the one designated to the local file management. In this class we find all the function to segment the data into chunks, read and write chunks and check the CRC. This class is called whenever a function needs to manage a chunk. The class also present a linked list of all the chunk the application owns. A specular list of all the missing chunks is also provided. They are used mainly by the NC algorithm, but are also used by the application as they has to be transmitted to the neighbors periodically. A *RandomChunk* list is compiled when the application starts: the aim of this list is to provide random chunk selection. The list is compiled at the start time to improve the performance during the execution.

Internet Receiver

The *Internet Receiver* class (Figure 8) is responsible of the Remote Interface (RI) with the cooperation server. This class, in cooperation with the *Internet Downloader*, manage the connection to the remote cooperation server, the reception of the chunks through this interface and the acknowledging process to the server. The *Internet Receiver* class is called by the *Internet Downloader* every time it receives a packet from the cooperation server. The *Receiver* class implements an interface of the *Downloader* to manage the received content. Into the *Receiver* class is implemented the logical algorithm to execute the received command, such as the call to the *FileSave* class to save the received chunk or to check which chunk to require to the cooperation server. A *Timer* instance is initialized to check for timeout errors. In case a timeout happens, the connection with the cooperation server is closed and re-initialized.

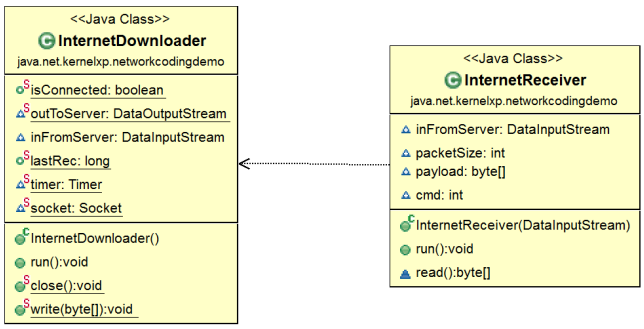


Figure 8: Internet Receiver class scheme

Broadcast Receiver

The *Broadcast Receiver* class (Figure 9) is developed following the same scheme as for the Internet Receiver and the Internet Downloader classes. The *Broadcast* class, in fact, defines a *Receiver* interface, implemented by the *BroadcastReceiver* class. In this structure the Broadcast class is responsible of the socket communication, since in this class are implemented the communication routines. The receiver decodes the received messages and make the action required by the received message. As a consequence, the *BroadcastReceiver* class will interact with a number of other classes, like the *FileSave* class with the aim of saving a received chunk, the *Devices* class to update the chunk status for the all the devices that belong to the cloud network. It also use the functions provided by the abstract NC class to communicate with the other neighbors, in a NC enabled environment.

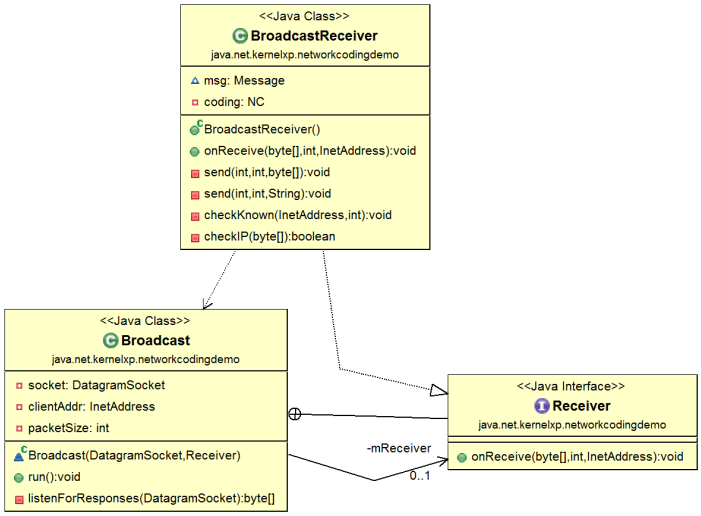


Figure 9: BroadCast Receiver class scheme

NC

This is the class (Figure 10) implementing the Network Coding algorithm. The abstract class *NC* implements the basic procedures and defines the basic functions. It contains a *Timer* instance to periodically publish updates about the chunk status. This abstract class can be extended to implement every algorithm. In this case we have implemented the two algorithms, RLB into the *LocalBroadcastNC* class and the 2-by-2 XOR into the *XORNC* class.

Each of the two class extends the methods defined into the abstract *NC* class, overriding the *saveChunk* and *sendChunk* functions. The XORNC class adds also some useful proprieties, like the *scoreMiss* and *scoreHave* vectors, that are used to count how many devices own any chunk, to find the best chunks to be transmitted.

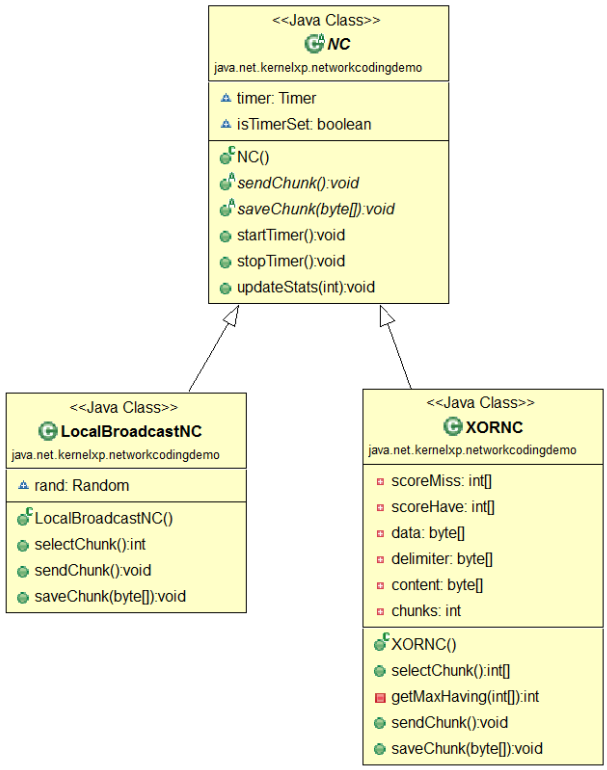


Figure 10: Network Coding class scheme

Utils

The *Utils* class (Figure 11) contains all the useful functions that are used across the whole application. It is declared as statical, as a consequence in any point of the application code is possible to access the functions contained into the *Utils* class. In this class il also stored the global variables, like the WifiP2PManager or the receivers for broadcast events.

The *ByteUtils* class, following the same concept, contains the functions to convert all the used data types into bytes sequence, to be transmitted over the network link. It also contains the inverse functions to decode byte sequences into any of the used data types.

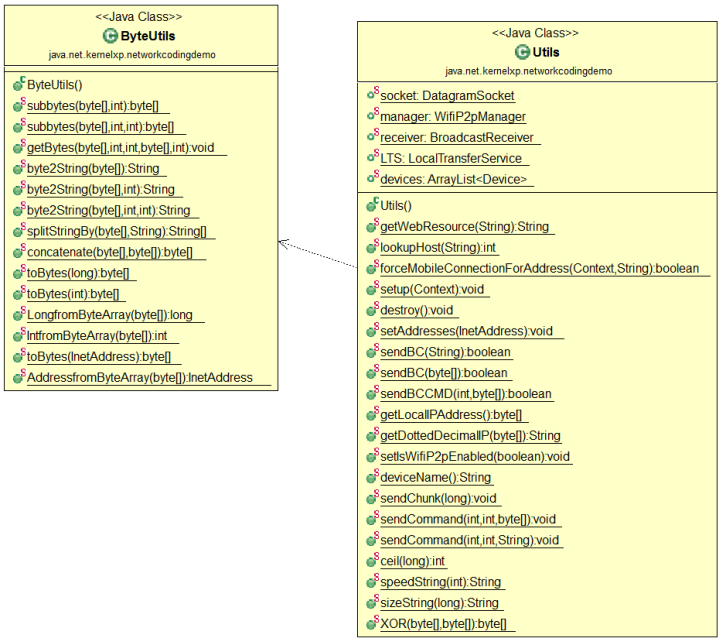


Figure 11: Utils Class scheme

WiFiDirectBroadcastReceiver

The *WiFiDirectBroadcastReceiver* class (Figure 12) is responsible of the P2P communication system. Indeed, this class contains all the functions used to search for Wi-Fi Direct enabled neighbors. The connection function and the role negotiating procedures are also implemented into this class. This way, accessing this class, is possible to control all the aspects of the P2P cloud sharing platform. When a new transmission is received, it is forwarded to the *Broadcast* class for managing.

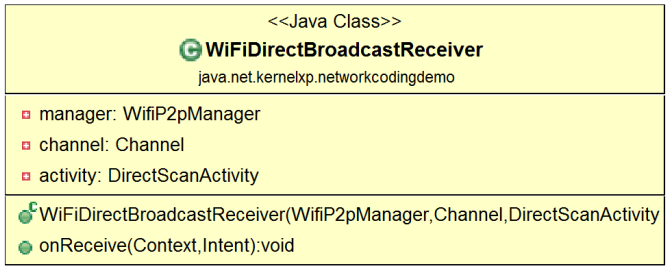


Figure 12: Wi-Fi Direct class scheme

Bibliography

[1] P. Pakzad, C. Fragouli and A. Shokrollahi, *Coding Schemes for Line Networks*,Laboratoire d'algorithmique (ALGO),Ecole Polytechnique Federale de Lausanne (EPFL) CH-1015 Lausanne, Switzerland

[2] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, Jon Crowcroft, *XORs in The Air: Practical Wireless Network Coding*,MIT CSAIL Univ. of Cambridge

[3] C. Fragouli,J. Le Boudec,J. Widmer, *Network Coding: An Instant Primer* in ACM SIGCOMM Computer Communication Review Volume 36 Issue 1, January 2006 Pages 63 - 68, 2006

[4] F. Fitzek, *Network Coding: Applications and Implementations on Mobile Devices*, Bodrum, Turkey. 2010.

[5] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla, *Code torrent: content distribution using network coding in vanet*, In MobiShare '06: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking, pages 1-5, New York, NY, USA, 2006. ACM.

[6] F. Fitzek and M. Katz, *Cooperation in Wireless Networks - Cooperation in Nature and Wireless Communications*,chapter 1, pages 1-27. Springer, 2006.

[7] F. Fitzek and M. Katz, *Cognitive Wireless Networks: Concepts, Methodologies and Visions Inspiring the Age of Enlightenment of Wireless Communications* ,Springer, July 2007

[8] M. Pedersen, F. Fitzek and T. Larsen, *Implementation and Performance Evaluation of Network Coding for Cooperative Mobile Devices* in Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference. Aalborg University, Denmark, 2008.

[9] D. Camps-Mur, A. Saavedra and P. Serrano, *Device to device communications with WiFi Direct: overview and experimentation*,Heidelberg, Germany, 2012.

[10] T. Ho, R. Koetter, M. Médard, D. R. Karger and M. Effros, *The Benefits of Coding over Routing in a Randomized Setting*,2003 IEEE International Symposium on Information Theory

[11] P. A. Chou, Y. Wu, and K. Jain, *Practical network coding*,Allerton Conference on Communication, Control, and Computing, Monticello,IL, 2003.

[12] S. Acedanski, S. Deb, M. Médard, R. Koetter, *How Good is Random Linear Coding Based Distributed Networked Storage?*, paper published during NetCod 2005

[13] C. Gkantsidis, J. Miller, P. Rodriguez, *Anatomy of a P2P Content Distribution system with Network Coding*, Microsoft Research, Cambridge.

[14] R. Jradi, L. Reedtz, *Ad-hoc network on Android*, Bachelor Thesis, Technical University of Denmark 2010.

[15] J. Ebrahimi and C. Fragouli, *Combinatiorial Algorithms for Wireless Information Flow*, EPFL Lausanne, Switzerland.

[16] J. Jou, M. Gerla, M.Stehr, *Content Network Coding on Androids: Energy Considerations*, DARPA CBMEN project

[17] L. Romera, *Location Based Pre-caching and Network Coding in Smart Content Distribution*,Master's Degree Project - January 2013

[18] L. Keller, A. Le, B. Cici, H. Seferoglu, C.Fragouli, A. Markopoulou, *MicroCast: Cooperative Video Streaming on Smartphones*, School of I&C EPFL, Lausanne, CH - 2012.

[19] M. Wang and B. Li, *Random push with random network coding in live Peer-to-Peer streaming*,IEEE Journal on Selected Areas in Communications, 25(9):1655–1666, Dec. 2007.

[20] M. Pedersen, J. Heide, F. Fitzek, and T. Larsen, *PictureViewer - a mobile application using network coding* In Proceedings of the 2009 European Wireless Conference, pages 151–156, May 2009.

[21] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath and C. A. Thekkath, *COMBINE: leveraging the power of wireless peers through collaborative downloading* In Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys), pages 286–298, 2007.

[22] R. Chandra, S. Karanth, T. Moscibroda, V. Navda, J. Padhye,R. Ramjee, and L. Ravindranath, *DirCast: a practical and efficient Wi-Fi multicast system* In Proceedings of the 17th IEEE International Conference on Network Protocols (ICNP), pages 161–170, Oct. 2009.

[23] H. Shojania and B. Li, *Random network coding on the iPhone: fact or fiction?* In Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), pages 37–42, 2009.

[24] M. Stiemerling and S. Kiesel, *A system for peer-to-peer video streaming in resource constrained mobile environments* In Proceedings of the 1st ACM Workshop on User-provided Networking: Challenges and Opportunities (U-NET), pages 25–30, 2009.

[25] S. Li and S. Chan, *BOPPER: wireless video broadcasting with Peer-to-Peer error recovery* In Proceedings of the 2007 IEEE International Conference on Multimedia and Expo,pages 392–395, July 2007.

[26] *WiFi Alliance. Wi-Fi direct*, http://www.wi-fi.org.