Evaluation of a content download service based on flute and LDPC for improving the quality of experience over multicast wireless networks

ISMAEL DE FEZ LAVA

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE COMUNICACIONES

EVALUATION OF A CONTENT DOWNLOAD SERVICE BASED ON FLUTE AND LDPC FOR IMPROVING THE QUALITY OF EXPERIENCE OVER MULTICAST WIRELESS NETWORKS

DOCTORAL THESIS

Ismael de Fez Lava

Supervisor:

Dr. Juan Carlos Guerri Cebollada

Valencia, Spain

December 2013

Collection doctoral thesis

© Ismael De Fez Lava

© 2014, of the present edition: Editorial Universitat Politècnica de València
Telf.: 963 877 012 / www.lalibreria.upv.es

ISBN: 978-84-9048-225-4 (CD version)

Abstract

This thesis dissertation studies file distribution in wireless networks, analyzing different mechanisms that allow to optimize the transmission in terms of bandwidth and Quality of Experience. Specifically, the thesis focuses on file transmission in multicast channels. Multicast file transmission results appropriate in certain environments and has several applications, some of them are presented in this work.

The thesis analyzes in depth FLUTE (File Delivery over Unidirectional Transport), a protocol for the reliable delivery of files in unidirectional channels, and presents some proposals to improve the transmission through FLUTE. In this sense, one of the basis of this protocol is the use of a mechanism called File Delivery Table (FDT), used to describe the files transmitted. This dissertation assesses how the transmission of the FDT affects the performance of the FLUTE protocol, and provides a methodology to optimize the content delivery.

On the other hand, in multicast file transmission services reliability is an essential premise. Among the mechanisms used by FLUTE to provide reliability, this work mainly focuses on AL-FEC (Application Layer – Forward Error Correction) codes, which add redundancy to the transmission in order to minimize the effect of the channel losses. Specifically, LDPC (Low Density Parity Check) codes are studied. The thesis evaluates LDPC Staircase and LDPC Triangle codes, comparing their performance under several transmission conditions.

Furthermore, in the case of having a feedback channel, one of the main contributions of this thesis is the proposal of adaptive LDPC codes for file download services. In these codes, the content server changes dynamically the amount of FEC protection provided depending on the losses detected by the users. The evaluation proves the good performance of these codes for different environments.

Resumen

Esta tesis estudia la distribución de ficheros en redes inalámbricas, analizando diferentes mecanismos que permiten optimizar la transmisión en términos de ancho de banda y calidad de experiencia. Concretamente, la tesis se centra en la transmisión de ficheros en canales multicast. Dicha transmisión resulta adecuada en ciertos entornos y tiene múltiples aplicaciones, algunas de las cuales se presentan en este trabajo.

La tesis analiza en profundidad FLUTE (File Delivery over Unidirectional Transport), un protocolo para el envío fiable de ficheros en canales unidireccionales, y presenta algunas propuestas para mejorar la transmisión a través de dicho protocolo. En este sentido, una de las bases de este protocolo es el uso de un mecanismo llamado Tabla de Envío de Ficheros (FDT), que se utiliza para describir los contenidos transmitidos. Este trabajo analiza cómo la transmisión de la FDT afecta al funcionamiento del protocolo FLUTE, y proporciona una metodología para optimizar el envío de contenido mediante FLUTE.

Por otro lado, en la transmisión de ficheros por multicast resulta esencial ofrecer un servicio fiable. Entre los distintos mecanismos utilizados por FLUTE para ofrecer fiabilidad, este trabajo analiza principalmente los códigos de corrección AL-FEC (Application Layer – Forward Error Correction), los cuales añaden redundancia a la transmisión para minimizar los efectos de las pérdidas en el canal. Al respecto, esta tesis evalúa los códigos LDPC Staircase y LDPC Triangle, comparando su funcionamiento bajo diferentes condiciones de transmisión.

Además, en el caso de tener un canal de retorno, una de las principales contribuciones de esta tesis es la propuesta de códigos LDPC adaptativos para servicios de descarga de ficheros. En esta clase de códigos, el servidor de contenidos cambia dinámicamente la cantidad de protección FEC proporcionada en función de las pérdidas que detectan los usuarios. La evaluación demuestra el buen funcionamiento de estos códigos en distintos entornos.

Resum

Esta tesi estudia la distribució de fitxers en xarxes sense fil, analitzant diferents mecanismes que permeten optimitzar la transmissió en termes d'amplada de banda i qualitat d'experiència. Concretament, la tesi se centra en la transmissió de fitxers en canals multicast. La transmissió multicast resulta adequada en certs entorns i té múltiples aplicacions, algunes de les quals es presenten en este treball.

La tesi analitza en profunditat FLUTE (File Delivery over Unidirectional Transport), un protocol per a l'enviament fiable de fitxers en canals unidireccionals, i presenta algunes propostes per a millorar la transmissió a través de FLUTE. En este sentit, una de les bases d'este protocol és l'ús d'un mecanisme anomenat Taula d'Enviament de Fitxers (FDT), que s'utilitza per a descriure els continguts transmesos. Este treball analitza com la transmissió de la FDT afecta el funcionament del protocol FLUTE, i proporciona una metodologia per a optimitzar l'enviament de contingut mitjançant FLUTE.

D'altra banda, en la transmissió de fitxers per multicast resulta essencial oferir un servici fiable. Entre el distints mecanismes utilitzats per FLUTE per a oferir fiabilitat, este treball analitza principalment els codis de correcció AL-FEC (Application Layer – Forward Error Correction), els quals afegeixen redundància a la transmissió amb l'objectiu de minimitzar els efectes de les pèrdues en el canal. En este sentit, esta tesis avalua els codis LDPC Staircase and LDPC Triangle, comparant el seu funcionament davall diferents condicions de transmissió.

A més, en el cas de tindre un canal de retorn, una de les principals contribucions d'esta tesi és la proposta de codis LDPC adaptatius per a servicis de descàrrega de fitxers. En esta classe de codis, el servidor de continguts canvia dinàmicament la quantitat de protecció FEC proporcionada en funció de les pèrdues que detecten els usuaris. L'avaluació demostra el bon funcionament d'estos codis per a distints entorns.

Agradecimientos

Quisiera aprovechar este espacio para expresar mi gratitud a la gente que ha hecho posible que esté escribiendo estas líneas.

En primer lugar, quisiera dar las gracias a Juan Carlos Guerri, director de esta tesis, por guiarme durante esta andadura. Gracias también a todos los compañeros del Grupo de Comunicaciones Multimedia que me han acompañado durante todos estos años. Especialmente me gustaría dar las gracias a Francisco Fraile, Román Belda y Pau Arce, por toda la ayuda, por todo lo que he aprendido con ellos y por los buenos momentos vividos desde que llegué al grupo. Mil gracias. Sin vosotros, este camino hubiera sido intransitable.

Gracias a todos mis amigos. En especial tengo que nombrar a Francisco, María, Sandra y Miriam, que están conmigo desde hace muchos años.

Gracias a mi familia, a los que están y ya no están entre nosotros. Gracias a Vicente, que forma parte de ella desde hace años. Gracias a Javi, a Dani y al que viene en camino, por toda la alegría y ganas de vivir que transmiten. Gracias a mi hermana Patricia, por todos los momentos que hemos compartido. No he podido tener mejor espejo en el que mirarme.

Y por último, gracias a mis padres, por tantas y tantas cosas. Porque estos años de esfuerzo que han supuesto esta tesis no son nada comparados con todos los años de esfuerzo por su parte para darnos a mi hermana y a mí lo mejor. Estoy orgulloso de vosotros. Os quiero. Esta tesis va por vosotros.

This is just the beginning...

Contents

List of Tables

List of Figures

Acronyms

3GPP	3rd Generation Partnership Project
ALC	Asynchronous Layered Coding
AL-FEC	Application Layer – Forward Error Correction
ARQ	Automatic Repeat Request
AVC	Advanced Video Coding
CCI	Congestion Control Information
CDS	Content Download Services
DASH	Dynamic Adaptive Streaming over HTTP
DTN	Delay-Tolerant Networking
DVB	Digital Video Broadcasting
DVB-H	Digital Video Broadcasting – Handheld
eMBMS	Evolved Multimedia Broadcast and Multicast Services
ESG	Electronic Service Guide
ESI	Encoding Symbol Identifier
FDT	File Delivery Table
FEC	Forward Error Correction
FLUTE	File Delivery over Unidirectional Transport
FTI	FEC Object Transmission Information
GE	Gaussian Elimination
GF	Galois Field
GOE	Generalized Object Encoding
GoP	Group of Pictures
GPS	Global Positioning System
HEL	Header Extension Length
HET	Header Extension Type
HSPA	High-Speed Packet Access
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force

IP	Internet Protocol
IPTV	Internet Protocol TV
ISO	International Organization for Standardization
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
LCT	Layered Coding Transport
LDPC	Low Density Parity Check
LT	Luby Transform
LTE	Long Term Evolution
MBMS	Multimedia Broadcast and Multicast Services
MD5	Message-Digest Algorithm 5
MDS	Maximum Distance Separable
MIME	Multipurpose Internet Mail Extensions
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
MTU	Maximum Transfer Unit
OTI	Object Transmission Information
PRR	Packet Reception Ratio
PSI	Protocol-Specific Indication
QoE	Quality of Experience
RFC	Request For Comments
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
SAP	Session Announcement Protocol
SBN	Source Block Number
SDP	Session Description Protocol
TCP	Transmission Control Protocol
TESLA	Timed Efficient Stream Loss-Tolerant Authentication
TOI	Transport Object Identifier
TSI	Transport Session Identifier
UDP	User Datagram Protocol
UEP	Unequal Erasure Protection
URI	Uniform Resource Identifier
WEBRC	Wave and Equation Based Rate Control
WiMAX	Worldwide Interoperability for Microwave Access
XML	Extensible Markup Language

Chapter 1

Introduction

This chapter introduces this thesis dissertation, presenting the scope of the thesis, as well as its objectives and contributions. Also, this chapter presents the structure of this thesis, introducing each one of the six chapters that compose this work.

1.1. Scope

Wireless networks are part of our daily lives for many years. Since the first half of the last century, people listened to the news or music through the waves that arrived to their radios. Some decades later, the use of the television became popular in all households. Today, former analogical broadcasting systems have been replaced by modern digital platforms, such as DVB (Digital Video Broadcasting) technologies or connected TVs.

In contrast to radio and television, other technologies were intended for wired scenarios and have evolved to wireless systems. There are two representative examples: telephone and Internet. Nowadays, approximately three quarters of the world population has access to a mobile phone [1]. This percentage exceeds the 90% in developed countries. In the last years, the appearance of smartphones has revolutionized the industry. Through these intelligent devices, users connect to the Internet using new technologies such as Wi-Fi, HSPA (High-Speed Packet Access), WiMAX (Worldwide Interoperability for Microwave Access) or LTE (Long Term Evolution).

In that regard, the Internet consumption through wireless networks has exploded in the last years due to the fact that not only the number of users has increased but also the amount of data consumed. Thus, the access to multimedia content has grown considerably and nowadays transmission networks carry all kind of multimedia traffic. The visualization of multimedia streaming through video portals and photo sharing on social networks are two good examples of the great importance that multimedia content has on existing networks. In this sense, a large number of the contents consumed by users require a high bandwidth.

In this way, currently, users demand greater access capacity to utilize an increasing number of services and applications. In turn, these applications become more and more hungry for bandwidth, especially video services and related applications. As a result, wireless broadband demand has experienced a hundredfold growth in the last years and one can only expect a similar expansion in the years to come.

In this framework, the exponential growth of the traffic in wireless networks is becoming a problem since, unfortunately, wireless broadband exploits a limited resource: frequency spectrum. For this reason, new wireless technologies improve spectrum efficiency and regulators reshape radio spectrum allocation, adapting it to satisfy the changing needs of society. However, this approach faces important challenges to cope with user demands. Modern wireless standards perform very close to theoretical limits. Without a major breakthrough, it is very unlikely that the efficiency of next generation telecommunication systems would be orders of magnitude greater than that of current technologies. Also, new wireless technology needs time to become established in markets and even more time to become economically profitable for operators. Policymakers face difficulties to reshape radio spectrum as any change requires extensive negotiations.

Therefore, as users require more bandwidth and the frequency spectrum is a limited resource, it is necessary to search for underexploited features of current wireless technologies that could optimize the usage of spectrum efficiency. Modern wireless standards provide wireless multicast access, which represents a good mechanism for reducing the bandwidth. Although nowadays a large proportion of the data traffic is sent through unicast IP networks, the use of multicast networks has a great importance too. For instance, multicast networks are used by TV channels or radio stations to broadcast their contents.

The utilization of multicast networks allows to send content to different users using a single transmission operation. Thus, depending on the service, multicasting results more efficient than unicasting, which generates more traffic in the network. In this way, multicast networks are highly recommendable when there is sufficient number of users interested in receiving the same contents.

Multicast networks are used to broadcast both video streaming and files. In this sense, the Quality of Experience (QoE) perceived by the users is an important parameter in the evaluation of both video streaming and file transmission services. In the first, the delay, the losses and the video quality are key aspects for the QoE. Regarding file transmissions, a good QoE is obtained when the file is received correctly with the minimum download time.

In file transmissions users need to receive correctly all the packets that compose a file in order to download it successfully, so channel losses should not occur. However, multicast transport does not guarantee, generally, error free communication and so it is needed to provide protection against errors. Additionally, some multicast file transmissions lack of a feedback channel that can be used by clients to report the servers about the status of their downloads. Therefore, clients are not able to ask for packets lost during the transmission. In this way, in the absence of retransmissions in the transport layer, it is necessary that multicast protocols provide reliability at the application layer to overcome possible packet loss in the communication channel. Among the existing protocols, one of the most used by the different standardization organisms is FLUTE [2]. FLUTE is highly used for delivering multimedia content in unidirectional environments in a reliable manner. One of the key elements of FLUTE is the use of a File Delivery Table (FDT), an in-band mechanism used to inform clients about the files (and their characteristics) transmitted within a FLUTE session. Clients need to receive the FDT in order to start downloading files. In this sense, the delivery of FDT packets and their proper configuration parameters have a great impact on the QoE of FLUTE services.

FLUTE provides reliability using different protection mechanisms. It should be noted that, generally, there are two main error correction techniques, ARQ (Automatic Repeat Request) and FEC (Forward Error Correction). The former consists of retransmitting data that are missed in the communication, whereas FEC allows to reconstruct the original data without retransmissions, through error correction encoding. FEC is mainly used in unidirectional environments, where a return channel does not exist.

In this sense, FLUTE works over a FEC block, which is used to protect the file delivery service. Although error correction is generally applied in the lower layers of a communication system, it can be used at higher layers. Specifically, AL-FEC (Application Layer FEC) provides additional robustness to certain services without any modification in the lower layers of a system, through applying FEC coding at transport packet level. Thus, the use of AL-FEC is particularly interesting for provisioning new services over communication networks already in place, since AL-FEC can increase the native reliability of the network to meet the requirements of a specific service, without additional infrastructure [3]. Moreover, AL-FEC may improve the performance of content transfer through wireless communication networks, as it can decrease download times as well as network traffic, since it avoids the request of lost packets.

The performance of AL-FEC is somewhat dependent on the complexity of the algorithm used to protect the information. There are different categories of FEC codes: convolutional codes, block codes, fountain codes and hybrid systems. In this sense, the most advanced algorithms fall in the category of rateless codes and perform very close to ideal FEC codes: no matter what is the erasure rate of the channel, receivers need only to acquire an amount of data equivalent to the size of the original file to be able to restore it. Nevertheless, rateless codes require more processing to generate the parity data for a specific file than other AL-FEC codes, such as LDPC (Low Density Parity

Check). In environments where the multicast content selection is dynamic, it may be impossible to generate the parity data and insert it in the network on time [4].

LDPC AL-FEC codes provide a good trade-off between performance (download time) and complexity (time required to generate parity and time required to do the decoding process) [5]. LDPC AL-FEC parity can be generated nearly in real time but, unlike rateless codes, the optimum code rate depends on the erasure rate of the channel. When the code rate of LDPC AL-FEC is adjusted to the actual loss rate of the channel, LDPC AL-FEC codes achieve a performance comparable to rateless codes in dynamic loss environments, especially when adaptive LDPC AL-FEC codes are used. Adaptive codes allow servers to send data at an optimum code rate for the channel losses experienced by the clients.

Moreover, in environments with limited bandwidth the performance of adaptive codes for file transmission can be improved by choosing the best protection that benefits the major part of users.

In this way, the use of file transmission services through multicast networks by employing an appropriate configuration can result very efficient, reducing considerably a resource so valuable as the bandwidth is but without degrading the Quality of Experience of users.

1.2. Objectives and contributions

The main objective of the thesis is:

> ***The study, analysis and development of a file transmission service for multicast wireless networks, based on the FLUTE protocol and LDPC AL-FEC codes, for improving the channel bandwidth utilization and the Quality of Experience perceived by the users.***

To achieve this objective, three different intermediate objectives or milestones are defined, which represent the main contributions of this thesis:

- **Objective 1**: **Analysis and evaluation of the FLUTE protocol, implementation of a FLUTE file delivery service and improvements to the protocol**

  First of all, an exhaustive study of the FLUTE protocol is carried out, analyzing its characteristics, its functioning and its applications. Moreover, a set of different use cases is defined and analyzed. To that extent, an implementation of a FLUTE file delivery service is carried out, developing both a FLUTE server and a FLUTE client, which is the starting point of the thesis. Based on this implementation, the main characteristics of FLUTE are analyzed, mainly the File Delivery Table, which is the most characteristic feature of the protocol.

In this sense, we propose certain modifications to the FDT that help to improve the QoE of users, such as mechanisms that reduce the download time and improve the personalization of file delivery services.

- **Objective 2: Analysis, implementation and evaluation of LDPC AL-FEC codes in unidirectional environments**

  Reliability is one of the basis of the FLUTE protocol, which employs different protection mechanisms, such as the use of AL-FEC codes. Among the different codes supported by FLUTE, this thesis is focused on LDPC codes. LDPC codes (both Staircase and Triangle structures) are analyzed, implemented and integrated into the developed FLUTE server and FLUTE client. A complete evaluation of LDPC in file delivery FLUTE sessions is carried out.

- **Objective 3: Analysis and evaluation of Adaptive LDPC AL-FEC codes for environments with a feedback channel**

  This thesis proposes adaptive LDPC codes for content download services. These codes are based on the use of a reporting mechanism through which the server obtains an estimation of the erasure rate perceived by every user. This information is used to send the information with an optimum protection that minimizes the download time on the clients side, thus improving the Quality of Experience of the users. The advantages of these codes are assessed, showing different studies that consider the main characteristics of file transmissions.

  Moreover, the thesis also presents an efficient proposal of adaptive LDPC codes for channels with limited bandwidth.

A detailed list of publications derived from this thesis is presented in the Appendix A.

1.3. Structure of the thesis

This thesis is divided into six chapters:

- **Chapter 1**: represents the introduction of this thesis dissertation, where the scope, objectives, contributions and structure of the thesis are presented.
- **Chapter 2**: presents the main technologies involved in this work. Two main technologies are analyzed: the FLUTE protocol and LDPC codes. The chapter explains the state of the art of these technologies, detailing the related work.

  Also, this chapter presents the implementation of the FLUTE server and client developed in the scope of this thesis. This implementation also includes a LDPC coding/decoding library.

  Finally, this chapter presents different use cases that show the functionality of file delivery services through multicast networks.

- **Chapter 3**: evaluates the two LDPC structures supported by FLUTE, that is, Staircase and Triangle. This chapter presents a complete evaluation of these codes, comparing its performance under different conditions, like channel losses or content sizes.
- **Chapter 4**: proposes the use of adaptive LDPC codes. In these codes, the server sends content with a certain protection according to the losses perceived by the clients of the file delivery service. The section presents a theoretical analysis of these codes, and evaluates their performance, comparing both analytical and experimental results.
- **Chapter 5**: represents an improvement of the codes presented in the previous chapter for limited-bandwidth environments. This chapter presents an algorithm that decides which is the optimum protection that benefits the major part of the users. Apart from a complete evaluation under different scenarios, this proposal is compared to the proposal of Chapter 4.
- **Chapter 6**: analyzes the importance of the File Delivery Table in FLUTE file delivery services. Specifically, this chapter assesses how the delivery frequency of the FDT affects the download time. Also, this chapter proves that the quality of a file delivery service based on the FLUTE protocol depends greatly on an appropriate configuration of AL-FEC codes and the FDT.

Chapter 2

State of the art and use cases

This chapter presents the state of the art of the main technologies involved in this thesis work. The chapter is mainly focused on the FLUTE protocol and LDPC codes. In this sense, this chapter also presents an implementation of a file transmission server and client based on FLUTE with LDPC codes developed in the context of this thesis, as well as different use cases referred to file distribution using FLUTE.

2.1. Introduction

As mentioned in the introductory chapter, the objective of this thesis is to develop a file download service to improve the channel bandwidth efficiency. In this regard, this proposal is based on multicast networks, where the bandwidth reduction comparing to unicast networks is proportional to the amount of users downloading contents. By using appropriate mechanisms to send the contents, the bandwidth could be consumed efficiently. Specifically, this thesis proposes the use of FLUTE and LDPC AL-FEC codes to send files in multicast wireless networks.

Therefore, it is interesting to start this thesis dissertation by analyzing in depth the main technologies used along this work. Thus, Section 2.2 presents the FLUTE protocol, detailing its characteristics and architecture, as well as the transmission process of this protocol. In this sense, the File Delivery Table, which is the main element of FLUTE, is analyzed in detail. Lastly, this section explains the different codes supported by FLUTE, providing an introduction to the following section, dedicated to LDPC codes.

Specifically, Section 2.3 studies the two LDPC structures defined in the FLUTE specifications, that is, LDPC Staircase and LDPC Triangle. The main characteristics of both structures and the transmission parameters are explained, by presenting the specifications of the RFC 5170 [6], where LDPC Staircase and Triangle are defined.

Once both FLUTE and LDPC has been explained, Section 2.4 analyzes the work related to these technologies.

On the other hand, Section 2.5 presents the implementation of the FLUTE server and client and LDPC codes developed in the context of this thesis. This implementation is used to carry the experimental evaluations presented in this dissertation.

Finally, Section 2.6 presents some use cases related to multicast file transmission, for instance personalized multimedia content distribution in shopping centers or touristic environments, or background push content download services for mobile devices.

2.2. FLUTE

2.2.1. Characteristics and architecture

FLUTE (File Delivery over Unidirectional Transport) is a protocol for the unidirectional delivery of files over the Internet, which is particularly suited to multicast networks. FLUTE was initially defined in RFC 3926 (October 2004) [7] and updated by the RFC 6726 (November 2012) [2]. Its main characteristic is that this protocol offers reliability in the transmission. Moreover, it provides massive scalability, management and congestion control.

FLUTE is applicable to the delivery of large and small files to many hosts. For instance, the protocol can be used for the delivery of large software updates to many hosts simultaneously. FLUTE could also be used for continuous, but segmented, data such as time-lined text for subtitling. Moreover, FLUTE is useful to send metadata, for example, Session Description Protocol (SDP) [8] files, or the Electronic Service Guide (ESG) in DVB-H (Digital Video Broadcasting – Handheld) [9].

The three main mechanisms used by FLUTE to provide reliability are: AL-FEC to add redundancy and correct errors; retransmissions by means of data carousels to receive packets previously lost; and offline file repair sessions to request certain packets that have not been received.

FLUTE file transfers are organized into file delivery sessions. A session is uniquely identified by the multicast source IP address and by a session identifier called TSI (Transport Session Identifier). Also, each session contains one or more associated channels, in which files are delivered. Each channel sends in a certain port number and with a given transmission rate. On the other hand, each file transmitted in a file delivery session is uniquely identified both by its content location and an internal identifier called TOI (Transport Object Identifier).

Fig. 2.1 shows the protocol stack of FLUTE. FLUTE works over Asynchronous Layered Coding (ALC) [10], the base protocol designed for massively scalable multicast distribution. At the same time, ALC uses the LCT (Layered Coding Transport) [11] building block for session management functionalities, the multiple rate congestion control (CC) building block, as well as the FEC building block [12] used for error control. In the lower layers, FLUTE works over UDP (User Datagram Protocol) on transport level and IP (Internet Protocol) on network level.

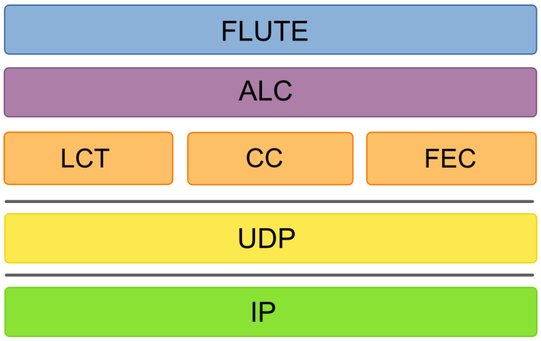


Fig. 2.1. FLUTE protocol stack.

On the other hand, to start receiving a file delivery session, clients need to know the transport parameters associated with the session. These parameters are obtained through the Session Description. This information can be sent out-of-band, through methods such as HTTP/MIME (Hypertext Transfer Protocol / Multipurpose Internet Mail Extensions) headers, XML (Extensible Markup Language) or SAP (Session Announcement Protocol). Normally, the Session Description uses the format defined by the SDP protocol [13]. Session Description must include the parameters that identify a session and a channel: source IP address, TSI and port number. Moreover, the Session Description can include additional information such as the number of channels or the congestion control algorithm used.

Once the clients have the necessary information to join a session and have established the connection, they can receive files. But before, they must know which files are being transmitted within the session and their characteristics. This information is obtained by means of the File Delivery Table, which is explained in detail in the following section.

2.2.2. FDT

The File Delivery Table (FDT) provides the means to describe various attributes associated with the files sent through the file delivery session. The FDT is described using XML language and is delivered through FDT Instances, which are FLUTE packets with a special LCT extension header (called EXT_FDT) used to indicate they carry FDT data. The XML is the payload of the packet to send. Also, the value 0 of TOI is reserved to identify a packet as FDT.

The FDT includes a mandatory attribute called “Expires” used to indicate the period of validity of the FDT. Also, regarding each file described in the FDT, the only two mandatory attributes in the description of each file are the content location (name, identification, and location of the file, which are specified as an URI –Uniform Resource Identifier) and the TOI.

The rest of attributes are optional, such as the content length, the content type or the content MD5 (Message-Digest Algorithm 5). Moreover, the FDT could carry the FEC Object Transmission Information, that is, information used to packetize a file in transmission and rebuild it in reception. The FEC Object Transmission information provides information like the size of each packet (parameter “FEC-OTI-Encoding-Symbol-Length”) or the type of FEC coding (parameter “FEC-OTI-FEC-Instance-ID”).

In this sense, the in-band delivery of the FEC Object Transmission Information is mandatory. If not included in the FDT, the server can make use of a LCT header extension called EXT_FTI (which is explained later). The FEC Object Transmission Information delivered to receivers must be exactly the same whether it is delivered using the FDT or using the EXT_FTI (or both).

Fig. 2.2 shows an example of an FDT. As the figure depicts, the FDT is composed of elements, each one containing different attributes. For instance, the description of the file with TOI = 1 includes the mandatory attributes “Content-Location” and “TOI” as well as the attribute “Content-Type”. On the other hand, the file with TOI = 2 includes also some attributes referred to the FEC Object Transmission Information (such as the “FEC-OTI-FEC-Instance-ID” or the “FEC-OTI-Maximum-Source-Block-Length”) and the attribute “Content-MD5”. A complete description of the elements and attributes in the FDT can be found in the RFC 6726 [2].

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:fl="http://www.comm.upv.es/flute"
     xsi:schemaLocation="http://www.comm.upv.es/flutefdt.xsd"
     Expires="2890842807">
     <File
        Content-Location="http://www.comm.upv.es/list.html"
        TOI="1"
        Content-Type="text/html"/>
     <File
        Content-Location="http://www.comm.upv.es/song2.mp3"
        TOI="2"
        Content-Length="612586"
        Content-Type="audio/mp3"
        FEC-OTI-FEC-Encoding-ID="3"
        FEC-OTI-Maximum-Source-Block-Length="64"
        FEC-OTI-Encoding-Symbol-Length="1428"
        FEC-OTI-Max-Number-of-Encoding-Symbols="1000"
        Content-MD5="+VP5IrWploFkZWc11iLDdA=="/>
</FDT-Instance>
```

Fig. 2.2. Example of an FDT.

The original FLUTE RFC [7] and the current FLUTE standard [2] establish that for every file delivered within a file delivery session, there must be a file description entry included in at least one FDT Instance sent within the session. A file description entry contains at a minimum the mapping between the TOI and the URI.

Also, both RFCs establish that the number of files described in each FDT is variable, that is, each FDT Instance contains at least a single file description entry and at most the complete FDT of the file delivery session. Therefore, there are two types of FDT Instances: partial FDT and complete FDT. An FDT Instance can be sent in any part of the file delivery session, so that FDT Instance packets are interleaved with data packets. Moreover, both RFCs indicate that the way FDT Instances are transmitted has a large impact on the transmission. It is recommended to repeatedly transmit FDT Instances describing files while these are being transmitted. Also, it is highly recommended to send the FDT Instances reliably using FEC. [2] suggests that mechanisms used for FDT Instances transmission should achieve higher delivery probability than the file recovery probability. Nevertheless, neither [7] nor [2] analyze how often an FDT Instance should be sent and how much FEC protection should be provided for each FDT Instance.

Precisely, this analysis is one of the main contributions of this thesis. An exhaustive study of the File Delivery Table is presented in Chapter 6.

2.2.3. FLUTE transmission

Fig. 2.3 shows a general overview of a file transmission using the FLUTE protocol. A FLUTE server has a repository of multimedia contents. For example, it could send video, audio, images, documents… The FLUTE server broadcasts the contents in a certain session (identified by the IP address and the TSI), which contains, at least, one delivery channel (identified by the port number). On the other hand, clients follow these steps:

- **Step 1**. Clients obtain through an out-band mechanism the Session Description that contains the transport parameters associated to the session. The way clients obtain the Session Description is independent of FLUTE.
- **Step 2**. Once the clients have connected to a certain session they have to wait until they receive the FDT that describes the files (and their corresponding metadata) that the server is sending.
- **Step 3**. Then, clients are able to identify the data packets they are receiving and they are able to download the files they are interested in.

If the server wants to stop sending data, the FLUTE protocol contains a field in its header to indicate clients that the session will be closed soon.

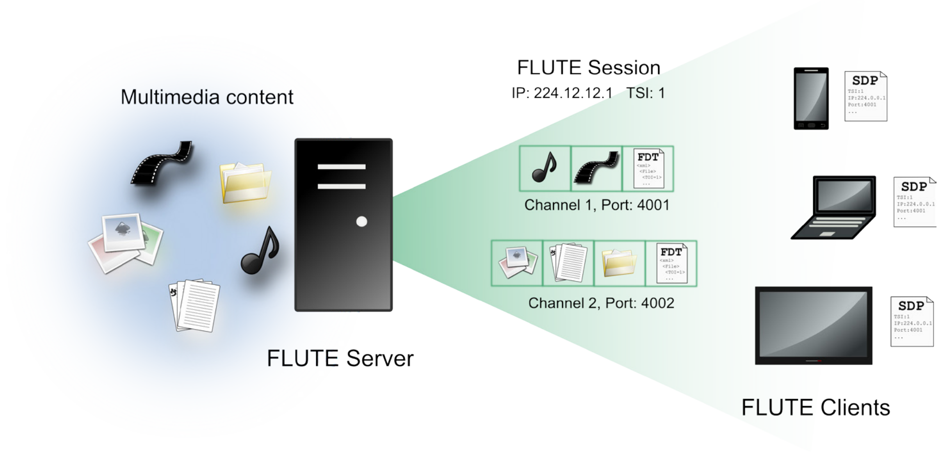


Fig. 2.3. File delivery using the FLUTE protocol.

Regarding the file transmission, in FLUTE each file represents a transport object. As Fig. 2.4 shows, each transport object is fragmented into source blocks. Also, each block is composed of encoding symbols. Segmentation of files is provided by a blocking algorithm (which calculates blocks from files) and a symbol encoding algorithm (which calculates encoding symbols from blocks). The algorithm used to split the file into blocks and symbols depends on the type of AL-FEC code used. In this sense, [7] proposes an algorithm that calculates source blocks from objects and encoding symbols from source blocks. The algorithm is very efficient, since the length of all source blocks is very similar, differing, at most, in one encoding symbol.

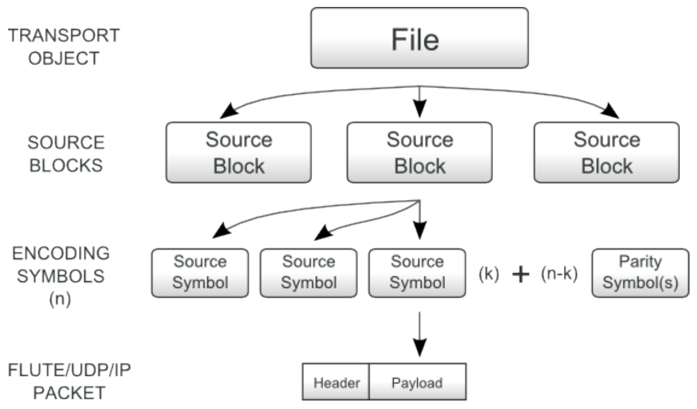


Fig. 2.4. FLUTE packet construction.

There are two types of encoding symbols: source and parity symbols. The former contain the original data of the file, whereas the parity symbols are created from a combination of source symbols (and other parity symbols), through FEC encoding, to provide reliability on the transmission. In this way, the generation of parity symbols depends on the type of codes used, as Section 2.2.5 and Section 2.3 explain. Thus, each block contains *n* encoding symbols, *k* of which are source symbols. If encoding is employed, the number of parity symbols per block is $n - k$. Generally, each symbol represents the payload of a FLUTE packet, although one FLUTE packet can contain several encoding symbols. The format of the FLUTE header is explained in the next subsection.

The values of *k* and *n* specify the code rate, the parameter used to determine the amount of protection provided to a file. The code rate is defined as k/n so, higher code rates imply less information protection. Obviously, when no AL-FEC is used $n = k$, so the code rate is 1.

In reception, clients are able to rebuild a file when they receive a number of packets equal to *k* * *inefficiency_ratio* [14]. The inefficiency ratio represents the relation between the number of packets needed to decode a file and the number of source packets that make up the file. The less the inefficiency ratio the more efficient is the coding (ideally this value is 1). The value of the inefficiency ratio depends on the coding algorithm. In codes that belong to the Maximum Distance Separable (MDS) category this value is equal to 1, whereas in the rest of codes the inefficiency ratio is greater than 1.

As for the organization of file transmissions, there are two different kinds of FLUTE delivery sessions: file transmission sessions and file carousels. In the latter, files are sent cyclically on a seamlessly endless loop. In this way, clients can complete their downloads if they have suffered losses in previous carousel cycles. Each carousel contains all the files to send. Furthermore, sessions can be static or dynamic, depending on whether the contents of the session change during its lifetime. The most used kind of sessions are file carousels. Recall that the use of carousels is one of the mechanisms used by FLUTE to provide reliability on the transmission.

2.2.4. FLUTE packet format

As mentioned when presenting the FLUTE protocol stack, both ALC and LCT building blocks provide basic transport to FLUTE, which inherits the requirements of these blocks. Therefore, the header of a FLUTE packet is composed by the headers of the protocols and building blocks that conform the FLUTE protocol stack. Thus, the format of a FLUTE packet is very similar to the format of an ALC packet. In fact, the format of FLUTE packets depends on the type of packets sent: data packets or FDT Instances. When data packets are sent the FLUTE packet format is equal to the ALC packet format, as Fig. 2.5 depicts.

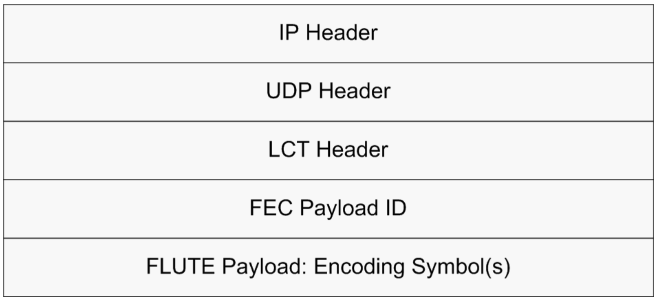


Fig. 2.5. FLUTE packet format for data packets.

On the other hand, the packet structure of FDT Instances is slightly different, since it includes an additional extension header (EXT_FDT) which indicates that this packet contains an FDT Instance, as Fig. 2.6 shows. Moreover, in FDT packets the value of the TOI is 0.

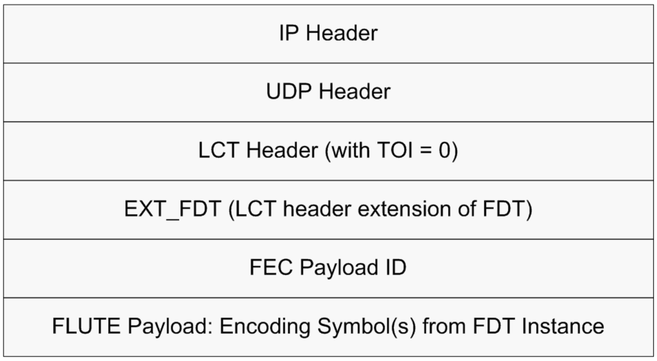

IP Header
UDP Header
LCT Header (with TOI = 0)
EXT_FDT (LCT header extension of FDT)
FEC Payload ID
FLUTE Payload: Encoding Symbol(s) from FDT Instance

Fig. 2.6. FLUTE packet format for FDT Instances.

The format of IP and UDP headers is well known. The structure of the LCT header is specified in RFC 5651 [11], depicted in Fig. 2.7:

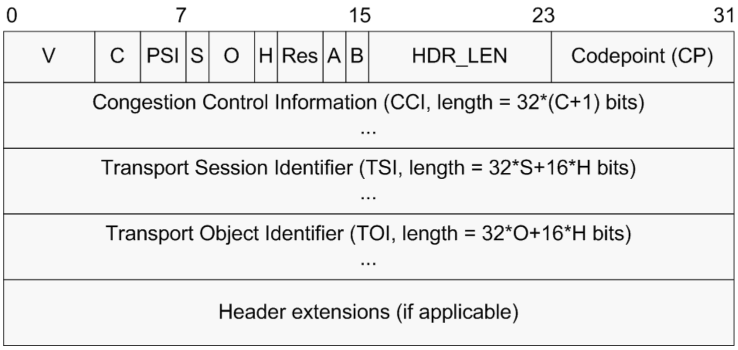

0
7
15
23
31
V
C
PSI
S
O
H
Res
A
B
HDR_LEN
Codepoint (CP)
Congestion Control Information (CCI, length = 32*(C+1) bits)
...
Transport Session Identifier (TSI, length = 32*S+16*H bits)
...
Transport Object Identifier (TOI, length = 32*O+16*H bits)
...
Header extensions (if applicable)

Fig. 2.7. Default LCT header format.

The meaning of each field is explained below:

- **V**: indicates the version. The latest version of the RFC of LCT [11] indicates that the value is 1.
- **C, S, O and H**: flags used to indicate the length of the fields CCI, TSI and TOI, as the figure shows.
- **PSI**: protocol-specific indication. In FLUTE these two bits are used for signaling purposes when AL-FEC mechanisms are used.
- **Res**: reserved.
- **A**: close session flag. It is used in order to indicate that the termination of transmission of packets for the session is imminent.
- **B**: close object flag. It indicates that the termination of transmission of packets for a certain object is imminent.
- **HDR_LEN**: LCT header length. It is the total length of the LCT header in units of 32-bits words.
- **Codepoint**: in FLUTE this field indicates the AL-FEC encoding used. It corresponds to the value of FEC Encoding ID.
- **CCI**: congestion control information. It is used to carry congestion control information, such as layer numbers, logical channel numbers or sequence numbers.
- **TSI**: transport session identifier. As mentioned, the TSI and the IP address uniquely identify the session.
- **TOI**: transport object identifier. It indicates to which object within the session this packet pertains. This value is equal to 0 for FDT Instances.
- **Header extensions**. Different header extensions have been defined:
  - **EXT_NOP**: non-operation extension.
  - **EXT_AUTH**: packet authentication extension. It includes information used to authenticate the sender of the packet.
  - **EXT_TIME**: time extension. It includes general purpose timing information.
  - **EXT_FTI**: FEC Object Transmission Information extension. This extension is intended to carry the FEC Object Transmission Information for an object in-band, that is, information related to the FEC encoding used. This information must be the same that the information included in the FDT (when the FDT includes information related to the encoding). The format of this field, therefore, depends on the type of AL-FEC codes used.
  - **EXT_CENC**: content encoding header extension. It is used to indicate the content encoding type when FDT Instances are coded. It supports following encoding algorithms: ZLIB [15], DEFLATE [16] and GZIP [17]. When EXT_CENC is used, it must be used together with a proper EXT_FDT.
  - **EXT_FDT**: it includes information about the FDT. This header extension is only included in packets that carry FDT Instances. The format of the EXT_FDT is shown in the following figure:

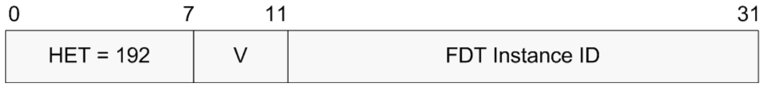


Fig. 2.8. EXT_FDT format.

The field "HET" indicates the type of LCT extension, whereas "V" refers to the version of FLUTE (the new RFC [2] indicates that this value must be 2). On the other hand, for each file delivery session, the numbering of FDT Instances starts from 0 and is incremented by one for each subsequent FDT Instance. Senders must not reuse an "FDT Instance ID" value that is already in use for a non-expired FDT Instance.

Back to the FLUTE packet format shown in Fig. 2.5 and Fig. 2.6, the FEC Payload ID [18] identifies uniquely a packet within the transmission of a certain packet. The FEC Payload ID is a 32-bit unsigned integer and it contains two fields:

- **Source Block Number (SBN)**: is used to identify from which source block of the object the encoding symbol in the payload of the packet is generated. The first source block is identified by the value 0, and the rest of blocks have a consecutive numeration.
- **Encoding Symbol ID (ESI)**: identifies which specific encoding symbol generated from the source block is carried in the packet payload. Again, the first ESI of a block is identified by the number 0 and so on.

It should be mentioned that the size of these two fields depends on the AL-FEC code used. As an example, Fig. 2.9 shows the structure of the FEC Payload ID for LDPC (Staircase and Triangle) structures.

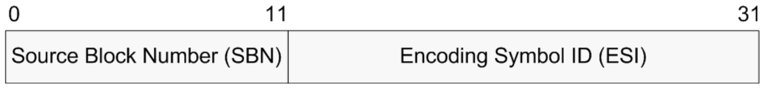


Fig. 2.9. FEC Payload ID encoding format for LDPC Staircase and Triangle.

Finally, coming back to the FLUTE protocol stack, through the congestion control building block, the server can send files in different channels with different code rates. In this way, different receivers joined to the same session may be receiving packets at different rates depending on the bandwidths of their individual connections to the sender. The RFC of ALC [10] indicates that implementations of ALC must support WEBRC (Wave and Equation Based Rate Control) [19], which provides rate and congestion control for data delivery. WEBRC is specifically designed to support protocols using IP multicast.

2.2.5. FEC codes

As explained, the format of some of the fields presented depends on the type of AL-FEC codes used. In this sense, FLUTE supports different codes. Each AL-FEC code is uniquely identified by an identifier called FEC Encoding ID. This identifier is included in the field "Codepoint" of the LCT header and optionally also is included in the FDT Instances.

The values of the FEC Encoding IDs assigned by the IANA (Internet Assigned Numbers Authority) [20] to identify the codes supported by FLUTE are:

0. Compact No-Code
1. Raptor
2. Reed-Solomon codes over GF (2^m)
3. LDPC Staircase codes
4. LDPC Triangle codes
5. Reed-Solomon codes over GF (2^8)
6. RaptorQ code

Both Compact No-Code and Reed Solomon codes belong to the Maximum Distance Separable category, where it is necessary to receive a number of encoding symbols equal to the number of packets that compose a file in order to rebuild it. In this way, both codes have an inefficiency ratio equal to 1. LDPC and Raptor codes have a value slightly higher than 1.

The use of one coding or another depends on the application but, in general, Raptor and RaptorQ work more efficiently at the expense of more complexity. Compact No-Code is recommended only on reliable channels (with very low losses), whereas Reed-Solomon is convenient when the amount of data to code is not very high. Also, LDPC codes are very efficient and their coding/decoding complexity is low.

Compact No-Code, Reed Solomon and Raptor (including RaptorQ) codes are briefly explained in the next subsections. Due to the relevance of LDPC codes in this thesis work, these codes will be explained in a separate section (Section 2.3).

2.2.5.a. Compact No-Code

Compact No-Code, specified by the RFC 5445 [18], does not apply any coding mechanism, i.e. only source packets are sent. Anyway, the RFC 5445 indicates the format of both the FEC Payload ID and the EXT_FTI. Both fields are necessary in order to divide an object in blocks and symbols, and to identify them. Since no parity symbols are generated, in reception it is needed to receive all the symbols that compose a file to download it.

2.2.5.b. Reed-Solomon

Reed-Solomon codes were invented by Irving S. Reed and G. Solomon in 1960 [21]. These codes are used for a lot of applications, such as data storage (for instance in Compact Discs), in wireless networks (mobile phones) or in satellite and wired communications, as well as in digital television (DVB uses Reed-Solomon to correct errors in the physical layer).

Reed-Solomon is an error corrector block code based on polynomials, and creates symbols by means of m-bits sequences. Each code word is composed of n symbols, which k are source symbols and r are parity symbols. The relation between the code word length and the number of symbols is defined by: $n = 2^m - 1$. These codes are able to correct up to $r/2$ erroneous symbols.

RFC 5510 [22] defines the FEC schemes for Reed-Solomon codes over GF (2^8) and over GF (2^m). In both cases, the creation of the n symbols through the k symbols that make a block is produced by means of a generation matrix. That matrix uses a polynomial which depends on the length of the m-finite field elements.

2.2.5.c. Raptor and RaptorQ

Raptor codes [23] were created in 2001 by Amin Shokrollahi. These codes belong to the fountain codes category, which allows to generate as many symbols as needed on the fly from the source symbols of a block, that is, a fixed code rate is not needed. Despite being a proprietary implementation, these codes have been adopted by several standardization organizations, such as IETF (Internet Engineering Task Force), 3GPP (3rd Generation Partnership Project) and DVB. Their main characteristic is that these codes are able to generate infinite parity information. Moreover, receivers need only few packets more than the number of packets that makes up the file for reconstructing it, regardless of the type of the received packets. That is the reason why these codes are very efficient. Furthermore, the encoding and decoding are very fast, so their implementation in software is easy.

The encoding process is divided in two steps: first, a precoding is done, in which l output packets are created through k input packets ($l>k$). The second step consist of the creation of the n source symbols from the l precoded symbols ($n>l$), using LT (Luby Transform) codes [24], a kind of fountain codes. Each symbol is generated independently, and it is possible to create an unlimited number of symbols. RFC 5053 [25] describes deeply the creation of the symbols and the header format related to Raptor.

On the other hand, in the last years a new generation of Raptor code has appeared, called RaptorQ. These codes provide better coding efficiency, since they recover missing data packets with minimal amounts of additional repair data (that is, a value of inefficiency ratio almost 1), at the expense of increasing the encoding and decoding complexity. Also, these codes support larger source symbol block sizes and provide superior flexibility. RaptorQ are defined in RFC 6330 (August 2011) [26].

2.3. LDPC (Staircase and Triangle)

2.3.1. Introduction

LDPC (Low Density Parity Check) codes were invented by Gallager in 1960 [27]. But they were not used until 30 years later, thanks to MacKay and Neal [28]. The original specification has suffered some improvements that make easy their utilization in different environments. For instance, they are the base of Tornado [29], LT and Raptor codes, all these proprietary implementations. LDPC belongs to the large block codes category, in which it is needed to receive more of the *k* packets that make up a file for reconstructing it. The codes included in this category are advisable when large files are encoded, since computational cost does not grow excessively.

Low Density Parity Check codes are systematic lineal block codes based on a parity check matrix used in the encoding and decoding processes. This matrix defines the relations between the different encoding symbols (source symbols and parity symbols). The matrix consist of some elements with values 0 and 1, and it is by definition disperse, since the most part of the elements are 0. By means of the matrix, the encoder generates the parity symbols through XOR operations on the source symbols and other parity symbols previously generated. Similarly, receivers use the matrix to reconstruct the symbols that have not been received by performing XOR operations on the encoding symbols already received. Fig. 2.10 shows an example of a parity matrix, and establishes the relations between source and parity symbols.

$$
(H_l|H_r) = \left(\begin{array}{cccccc|ccccc}
s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & p_6 & p_7 & p_8 & p_9 & p_{10} \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
\multicolumn{6}{c|}{\xleftrightarrow{\quad}} & \multicolumn{5}{c}{\xleftrightarrow{\quad}} \\
\multicolumn{6}{c|}{k} & \multicolumn{5}{c}{n\text{-}k}
\end{array}\right)
\qquad
\begin{array}{l}
p_6 = s_1 \oplus s_3 \oplus s_4 \oplus s_5 \\
p_7 = s_0 \oplus s_1 \oplus s_2 \oplus s_5 \\
p_8 = s_0 \oplus s_2 \oplus s_3 \oplus s_4 \\
p_9 = s_0 \oplus s_1 \oplus s_4 \\
p_{10} = s_2 \oplus s_3 \oplus s_5 \\
\xleftrightarrow{\qquad\qquad\qquad} \\
n\text{-}k \text{ equations}
\end{array}
$$

Fig. 2.10. Example of an LDPC parity check matrix (k = 6, n = 11).

The figure depicts a matrix with values k = 6 and n = 11, which generates 5 parity symbols per block. The size of the matrix is [(n - k) x n], so there are $n - k$ rows, each one representing an equation. The columns are related to the symbols of the block. Each element of the matrix with the value 1 (h_{ij} = 1) indicates that the j-th symbol takes part in the i-th equation. Thus, for instance, the first parity symbol (identified as p_6) is composed of the XOR sum of the symbols s_1, s_3, s_4 and s_5. Receivers are able to recover a symbol from an equation once they have successfully received all other symbols that take part in the given equation.

Moreover, a parity symbol can take part in the creation of other parity symbols. In general, each source symbol takes part in a fixed number of equations, that is, the number of 1s that contains the corresponding column. That parameter is called *N1*. The number of non-null elements of a row or column is called degree.

On the other hand, the matrix is divided into two sub-matrixes: the left and the right sub-matrixes. The first refers to source symbols, whereas the right sub-matrix refers to parity symbols. Obviously, receivers must use the same parity matrix as the sender in order to successfully decode each source block. Sender and receivers obtain the parity check matrix via a predefined algorithm (depending on the type of LDPC structure). The algorithm generates the matrix using these input parameters: number of source symbols (k), number of encoding symbols (n), number of equations to which a source symbol belongs to (*N1*) and seed used to generate the pseudorandom numbers. The sender signals all these parameters in the EXT_FTI header extension of LDPC so that receivers can generate the exact same matrix used by the encoder.

Depending on the parity check matrix structure there are two kinds of LDPC codes: regular codes and irregular codes. In the first ones, all the rows of the matrix have the same degree and all the columns have the same *N1* value, while irregular LDPC codes do not fulfill either condition. Gallager and Mackay codes are example of LDPC regular codes, whereas LDPC Staircase and Triangle are irregular codes.

In this sense, FLUTE supports two specific structures of LDPC: Staircase and Triangle. These structures only differ in the right sub-matrix generation: one has a shape like a staircase, and the other like a triangle, as Fig. 2.11 shows. In the LDPC Triangle structure, the degree of each row is equal or higher than that of the LDPC Staircase structure.

$$
(H_i | SC_5) = \left(\begin{array}{cccccc|ccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1
\end{array} \right)
\qquad
(H_i | Tr_5) = \left(\begin{array}{cccccc|ccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1
\end{array} \right)
$$

Fig. 2.11. Example of LDPC Staircase and LDPC Triangle matrix ($k = 6$, $n = 11$).

2.3.2. RFC 5170 specifications

The RFC 5170 [6], from June 2008, introduces the LDPC-Staircase FEC codes and the LDPC-Triangle FEC codes. Both schemes belong to the broad class of large block codes, according to the definition of the RFC of the FEC building block [12].

The RFC 5170 defines the parity matrix generation in both structures. With this purpose, it provides an algorithm that creates the parity matrix using certain input parameters. In both schemes, the algorithm is the same for the left sub-matrix but different for the right sub-matrix. For the creation of the matrix, the RFC proposes the use of the pseudorandom number generator algorithm of Park-Miller [30]. The RFC, as well, defines the fields of the LCT header extension EXT_FTI for LDPC, which includes the coding parameters, as Fig. 2.12 shows:

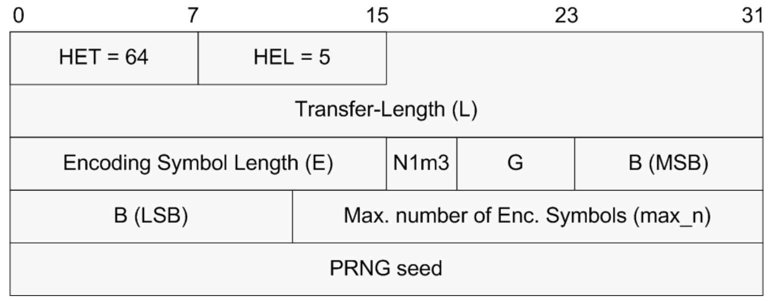


Fig. 2.12. EXT_FTI Header for LDPC Staircase and Triangle.

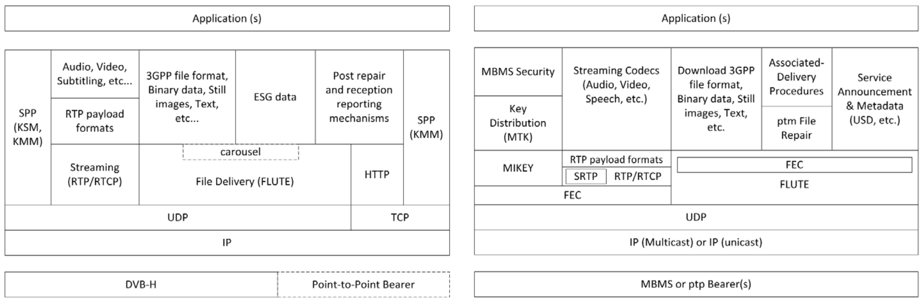
Following each field is briefly explained:

- **HET**: header extension type. It indicates the type of LCT header extension. EXT_FTI has a value of 64.
- **HEL**: header extension length, meaning the length of the whole header extension field, expressed in multiples of 32-bit words. In this case, this value is equal to 5.
- **Transfer length**: length of the transport object that carries the file in bytes. If no encoding is used, this value is equal to the file length.
- **Encoding Symbol Length (E)**: length of each encoding symbol in bytes, that is, the payload length of a FLUTE packet.
- **N1m3**: represents the value *N1* minus 3. Since the recommended value of *N1* is 3 (according to the RFC 5170), the recommended and default value of *N1m3* is 0, although it could have values from 0 to 7.
- **G**: number of encoding symbols per group. If the value of this field is 1, it means that each packet contains exactly one symbol.
- **B**: maximum source block length. It indicates the maximum number of encoding symbols generated for any source block. This field is split into two parts: the most significant bits (MSB) and the least significant bits (LSB).
- **max_n**: maximum number of encoding symbols generated for any source block.
- **PRNG seed**: this value is used to initialize the pseudorandom number generator.

Regarding the decoding process, decoding consists in solving a system of $n - k$ linear equations whose variables are the n encoding symbols. There are different algorithms used to resolve this system. One of the simplest is the algorithm proposed in [31], where given a set of linear equations, if one of them has only one remaining unknown variable, then the value of this variable is that of the constant term. So, this variable is replaced in all the remaining linear equations and reiterate. Thus, the value of several variables can be found recursively. Another decoding algorithms are based on Gaussian elimination, which is more efficient (since it rebuilds the file using less encoding symbols) at the expense of more complexity and thus processing.

2.4. Related work

The FLUTE protocol has been established as the multicast file delivery protocol for different standards, such as DVB-H [9] and DVB-IPTV (DVB – Internet Protocol TV) [32]. Also FLUTE is used by the 3GPP Multimedia Broadcast Multicast Service (MBMS) [33] and evolved MBMS (eMBMS) [34] to download multimedia content. Fig. 2.13(a-b) show, as an example, the protocol stack of DVB-H and MBMS, respectively. As mentioned, both technologies use FLUTE to send different types of files, such as multimedia files, metadata or the Electronic Service Guide in DVB.


Application (s)
SPP (KSM, KMM)
Audio, Video, Subtitling, etc...
RTP payload formats
3GPP file format, Binary data, Still images, Text, etc...
ESG data
Post repair and reception reporting mechanisms
SPP (KMM)
carousel
Streaming (RTP/RTCP)
File Delivery (FLUTE)
HTTP
UDP
TCP
IP
DVB-H
Point-to-Point Bearer
Application (s)
MBMS Security
Key Distribution (MTK)
Streaming Codecs (Audio, Video, Speech, etc.)
Download 3GPP file format, Binary data, Still images, Text, etc.
Associated-Delivery Procedures
ptm File Repair
Service Announcement & Metadata (USD, etc.)
MIKEY
RTP payload formats
SRTP
RTP/RTCP
FEC
FEC
FLUTE
UDP
IP (Multicast) or IP (unicast)
MBMS or ptp Bearer(s)

(a) IP datacast over DVB-H (b) MBMS IP multicast

Fig. 2.13. Protocol stack of DVB-H and MBMS.

In this regard there are research works that analyze the use of FLUTE in DVB and MBMS. For instance, [35] presents FLUTE as a multicast content delivery protocol to be used by DVB-H and MBMS, making a comparison among different transmission protocols. On the other hand, [36] proposes a hybrid transmission network where FLUTE is used by DVB-H to broadcast multimedia content, whereas [37] develops a portable middleware for DVB-H clients. Also, other papers explore the use of FLUTE in MBMS, for example [38].

But the use of FLUTE is not limited only for DVB and MBMS. For instance, [39] proposes the integration of FLUTE and TESLA (Timed Efficient Stream Loss-Tolerant Authentication) over satellite networks, whereas [40] presents an architecture for scalable DTN (Delay-Tolerant Networking) communication in sparsely populated areas based on FLUTE, similar to that proposed in [41].

Apart from file distribution, other papers propose FLUTE to provide video on demand services, such [42]. Moreover, [43] introduces an efficient progressive downloading over multimedia broadcast multicast service using FLUTE. Another related solution to highlight is [44], which analyzes on-demand video services using ALC.

Among the research works related to FLUTE, the paper that best analyzes the behavior of the protocol is, probably, [45]. This paper makes a complete analysis of FLUTE, evaluating the different configuration parameters of the protocol and how these parameters affect the transmission. Among the several contributions to the FLUTE protocol from the authors of [45], one of the most important has been the publication of an open source implementation of FLUTE, available in [46]. In fact, that implementation has been used in different research works, for instance in some papers that analyze AL-FEC codes. Other research works of these authors worth highlighting are: [47], which presents a bandwidth-efficient file delivery system using the server file format for FLUTE; [48], where the use of congestion control protocols in FLUTE is studied; and [49], which analyzes the impact of packet scheduling and packet loss distributions on FEC performances for content broadcasting applications. Regarding

LDPC, these authors have also published an open source implementation, in C language, of LDPC Staircase and LDPC Triangle codes for FLUTE applications [50].

In this sense, there are several works focused on LDPC codes, their efficiency, their applications and the proposal of new codes based on LDPC. In the context of this thesis, a complete comparison between the two LDPC structures supported by FLUTE (Staircase and Triangle) can be found in [51], which analyzes, among other parameters, the inefficiency ratio and the encoding/decoding time, typical evaluation parameters of error protection codes.

Other studies are focused on the efficiency of the decoding process, since the algorithm used is a key factor that affects the decoding efficiency and the energy consumption on the receiver. For instance, [52] presents low-complexity LDPC codes using maximum likelihood decoding. On the other hand, [53] and [54] show how the use of a hybrid Zyablov iterative decoding/Gaussian elimination (ID/GE) scheme can improve the erasure recovery capabilities of LDPC Staircase and Triangle codes, approaching the performances of ideal codes. Another paper that presents an optimization of LDPC codes in terms of efficiency is [55].

A new proposal to highlight is [56], based on LDPC Staircase codes, which proposes a Generalized Object Encoding (GOE) FEC scheme for the protection of one or multiple objects. This solution enables an Unequal Erasure Protection (UEP) of different portions of a given object and an efficient and global protection of a set of potentially small files.

2.5. Implementation

2.5.1. Introduction

In the context of this thesis, an own implementation of a FLUTE server and a FLUTE client has been developed. This implementation is used in order to carry out the different evaluations presented in this thesis work.

The first version of the implementation fulfilled the requirements of the RFC of FLUTE 3926 (version 1) [7] and the RFC of LDPC Staircase and Triangle [6]. This implementation was updated with the new version of FLUTE (defined in RFC 6726 [2]).

As we have mentioned in the related work, there are two open source libraries available on the Internet that implement the FLUTE protocol [46] and LDPC (Staircase and Triangle) codes [50]. In contrast to these libraries, the implementation hereby presented is specifically developed for mobile devices, and it has been programmed using Java language instead C.

Following subsections detail the file transmission server and the file transmission receiver developed, both based on FLUTE with LDPC (Staircase and Triangle) codes.

2.5.2. FLUTE server

Fig. 2.14 shows the architecture of the file download server implementation, based on the FLUTE protocol. The FLUTE sessions and channels management and their delivery through ALC protocol is done by means of the corresponding classes. Each "FluteSession" contains at least one "FluteChannel", which contains the different files sent during the session. Each file transmitted is represented by an object, represented by the class "FluteObject". Each time a "FluteObject" is inserted in a certain session, automatically the FDT is generated or updated. The class "FluteManager" is in charge of managing both sessions, channels and objects, and generating the FDT.

The packet delivery is carried out with a rate fixed by the class "RateControl", using a transmission model managed by the "Scheduler" block. Specifically, the class "RateControl" manages the transmission rate and the "Scheduler" is in charge of sending packets according to a specified policy. For instance, packets can be sent sequentially or randomly.

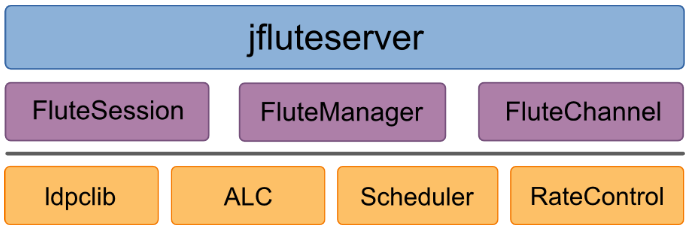


Fig. 2.14. File transmission server structure.

The "ldpclib" library implements the LDPC encoder and decoder (both schemes Staircase and Triangle). This library creates the parity matrix, which defines the relation between source and parity symbols. Also, the transmitter creates the header including the coding parameters. In this way, the receiver can generate the same parity matrix and do the decoding.

Regarding the "ldcplib" library, taking into account that the parity matrix is disperse, it has been developed a quadruple linked list for each row and column in order to reduce the memory consumption. Each non-null element of the parity matrix, represented by the class "LdpcEntry", points to its "LdpcEntry" neighbors (up, down, right, left). The use of this quadruple linked list, apart from reducing computational resources, simplifies the decoding process. On the other hand, in order to generate the parity matrix it is used a pseudo-random generator called Park-Miller minimal Standard (PRNG), as specified by the RFC 5170.

The library is built over Java SE (J2SE), so it can be used in any application that supports J2SE. The application allows to manage sessions and channels, configuring the corresponding parameters (IP Address, TSI, port and transmission rate), as well as adding different objects to each channel. Also, it is possible to choose for each object the coding used, specifying its code rate.

2.5.3. FLUTE client

The FLUTE client structure, shown in Fig. 2.15, is very similar to the structure of the server and shares most of the code.

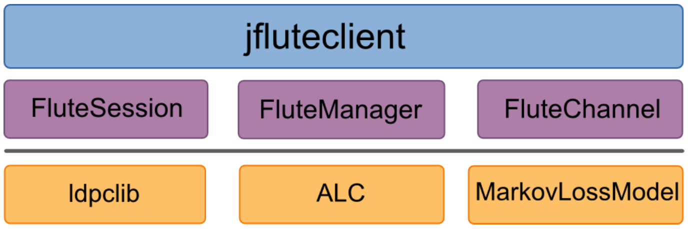


Fig. 2.15. File client structure.

The client is designed to support two different scenarios: mobile phone devices and PCs. This latter client is used to carry out the different evaluations, since it allows to manage the channel losses. In order to simulate these losses, in this thesis the two-state Markov model (also known as Gilbert model) has been used. We have chosen this model because it simulates well the burst losses (typical in wireless networks) and because it is widely used in literature [57]. In this way, our library includes a block that implements the two-state Markov loss model.

In our study, the decoding is performed using a simple iterative decoding algorithm, as the flow chart of Fig. 2.16 shows. When a new packet arrives, if the packet has not been previously received, the client obtains the parity matrix associated to the block that the symbol belongs to and checks the rows related to that particular symbol. In our algorithm, the decoding is based on partial sum buffers in each row of the parity matrix. Each buffer contains the XOR sum of the received symbols of a row. When all packets of a row except one have been received, the data of the non-received symbol is the partial sum of the buffer of that row. In this way, it is possible to reconstruct a symbol that has not been received yet.

As the case of the FLUTE server, the FLUTE client implementation is built over Java, therefore it can be used by any application that supports Java. Current technologies change very fast, so the obsolete implementation for J2ME has given way to a new application for Android devices.

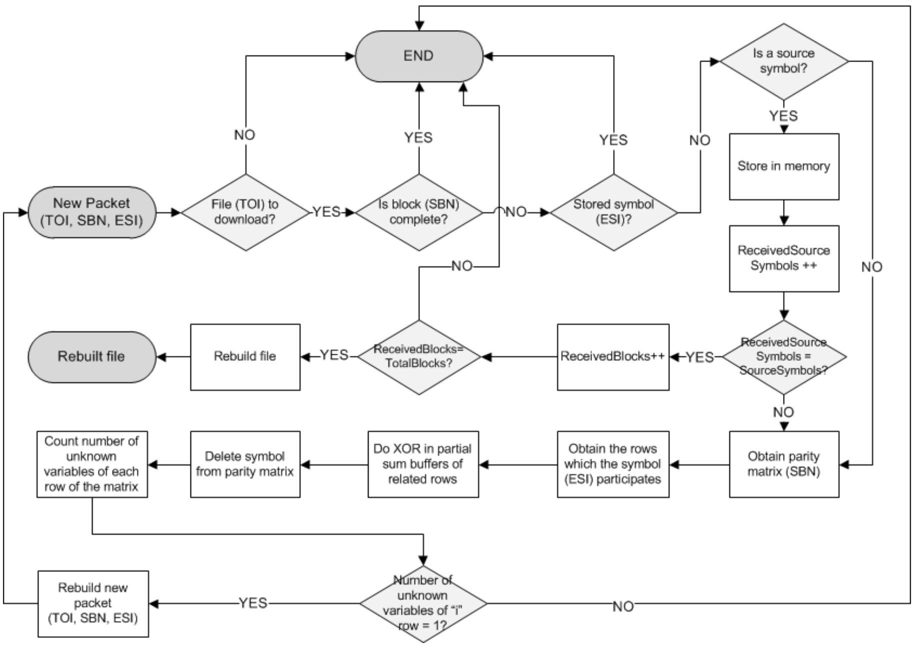

END
Is a source symbol?
YES
NO
Store in memory
ReceivedSource Symbols ++
New Packet (TOI, SBN, ESI)
File (TOI) to download?
Is block (SBN) complete?
Stored symbol (ESI)?
Rebuilt file
Rebuild file
ReceivedBlocks= TotalBlocks?
ReceivedBlocks++
ReceivedSource Symbols = SourceSymbols?
Count number of unknown variables of each row of the matrix
Delete symbol from parity matrix
Do XOR in partial sum buffers of related rows
Obtain the rows which the symbol (ESI) participates
Obtain parity matrix (SBN)
Rebuild new packet (TOI, SBN, ESI)
Number of unknown variables of "i" row = 1?

Fig. 2.16. Packet reception flow chart.

2.6. Use cases

2.6.1. Introduction

There are several environments where the use of multicast file delivery is very useful, mainly in environments where many users are interested in the same contents. In crowded locations (for example in some places of a city such as sights) users could benefit of receiving on their mobile devices different information (images, explanatory brochures, videos…) related to the place they are visiting. In these situations, it results interesting to send the same information (the same files) to a large number of people at the same time.

That is, multicast file transmission is very useful in crowded environments where users have similar interests. Other examples are cinemas or theatres, popular festivals and celebrations, or sport environments. In this sense, as an example of broadcast content distribution in crowded environments we highlight Cisco StadiumVision Mobile [58], a solution that delivers content to mobile devices in sports and entertainment venues.

A possible use case could be the transmission of different kinds of content in a basketball court: on the one hand the retransmission of the best moments of a match; on the other hand, information related to the shops within the court (burgers, clothes shops…) promoting their products and offering discount vouchers. This solution would benefit the three main actors implied: users would benefit of a better Quality of

Experience, shop owners would obtain more economic benefits due to the effect of advertisement, and the organization would increase its incomes (more users and more incomes from the shops).

On the other hand, the performance of systems based on file transmission to multiple users can be greatly improved by using recommendation mechanisms, which help to filter the most interesting contents for the user. The use of recommender systems (specifically content-filtering recommenders) is really useful in environments in which there is not a direct communication between the users and the server, such as in multicast networks. This is especially important when the amount of contents sent is rather high. In this sense, Section 2.6.2 is focused on personalization.

During the elaboration of this thesis the author has participated in different projects related to multicast content distribution, listed in the Appendix B. These projects represent good use cases that show the usefulness of the FLUTE protocol. Following sections present some of them. Specifically, Section 2.6.3 explains an architecture for content distribution in malls, Section 2.6.4 presents a personalized multimedia file distribution system for touristic environments, and Section 2.6.5 proposes background push content download services for the delivery of television programs.

Finally, Section 2.6.6 proposes the use of a hybrid system FLUTE/DASH to provide mobile video streaming services over broadcast wireless networks.

There are other use cases based on FLUTE in which the author of this thesis has participated. For instance, [36] presents a business model management platform for mobile multimedia delivery services through which service operators can enable several revenue streams simultaneously and users can access personalized content at an affordable cost [P.1]. On the other hand, [59] presents a multimedia on demand platform for emergency systems. The objective of this system is to offer a complete solution for providing multimedia services (video on demand and reliability file transfer) to a rescue team in an emergency situation.

2.6.2. Personalization

Personalization is one of the most important design aspects of mobile multimedia services [60]. Personalization provides many advantages to the user, for instance the automatic discovery of interesting content. The perception of the service adapting to the user preferences and needs encourages a positive experience, most significantly when accessed from personal devices such as mobile devices.

Personalization is also a great asset for content distributors. An efficient and functional application favors the consumption of content, thus increasing the market share of mobile content production. Since users receive content according to their preferences, content distributors can rely on the platform delivering the content only to their target audience and advertisers can improve the impact of their campaigns.

The information used by many recommender systems comes from two main sources: the user profile and the history. On the one hand, users can specify their preferences in their user applications, by indicating, for example, their favorite genre of film or their favorite actors. On the other hand, the application can learn the preferences of the user by analyzing their history, for instance, films the user has consumed. In this way, with the information indicated by the user and their consumption history, the application can recommend new films to the users, discarding several films that do not fit the preferences of the user.

Moreover, another interesting concept in personalization systems is the usage context. The concept of context refers to those information items that can be used to characterize the situation of entities (people, places or objects), which are relevant for the interaction among the user and the application [61]. In this sense, context-sensitive systems adapt their services according to the context information received, without any interaction from the user. To achieve this, it is desirable that both devices and services have access to context information and react accordingly. For instance, the user application detects (for example by means of GPS) that the user is in the cinema and automatically the mobile changes its state to vibration mode.

In the literature we can find hundreds of works related to personalization, for instance [62] and [63]. Also, several works deal with personalizing intelligent environments, such as [64]. Moreover, there are different proposals that use context information to improve the performance of content distribution systems. Thus, [65] presents a state of the art of context-sensitive systems. In the area of context-aware mobile interaction, several proposals have been carried out to adjust the mobile configuration profile according to changes in the user context [66]. Also, there are proposals that allow users to personalize the mobile interaction depending on the context information [67].

In the context of this thesis, regarding the FLUTE protocol, the personalization can be provided by using the File Delivery Table. Recall that the FDT carries metadata of the files that the FLUTE session is delivering. The structure of the FDT is flexible, according to the specifications of the RFC 6726. In this way, it is possible to add new elements and attributes to the FDT (by means of new labels) in order to include additional information regarding the contents sent. In fact, the standard DVB-H [9] proposes the use of an additional "group" field in the FLUTE FDT to enable logical grouping of related files. This is useful, for instance, in software update packages (which are usually composed of several files) or in web pages (which are usually linked to each other).

Following the example related to cinema, Fig. 2.17 shows an example of an FDT including additional metadata. As figure shows, each element "File" includes new elements referred to the genre, the director and the actresses of a film.

The problem of this structure is that there could be an infinite number of new elements, so receivers could have problems when parsing the FDT. In this thesis we propose the use of a general attribute called "Metadata", which contains all the additional metadata of the file in a format dependent on each particular application. Fig. 2.18 shows an example of the attribute proposed. In this way, only receiver applications interested in the metadata of the file would have to interpret these data.

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance
    xsi:schemaLocation="http://www.comm.upv.es/flutefdt.xsd"
    Expires="2890842807">
    <File
       Content-Location="http://www.comm.upv.es/killbill.mp4"
       TOI="1"
       Content-Type="video/mp4">
       <Genre> “Thriller” </Genre>
       <Director> “Quentin Tarantino” </Director>
       <Actress> “Uma Thurman” </Actress>
       <Actress> “Lucy Liu” </Actress>
    </File>
</FDT-Instance>
```

Fig. 2.17. Example of an FDT with metadata.

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance
    xsi:schemaLocation="http://www.comm.upv.es/flutefdt.xsd"
    Expires="2890842807">
    <File
       Content-Location="http://www.comm.upv.es/killbill.mp4"
       TOI="1"
       Content-Type="video/mp4">
       <Metadata> Director: Quentin Tarantino; Genre:
          Thriller; Actress: Uma Thurman, Lucy Liu
       </Metadata>
    </File>
</FDT-Instance>
```

Fig. 2.18. Example of an FDT with the proposed metadata structure.

Thus, the label “Metadata” could contain several attributes, and receivers could parse the metadata as long as they know the structure of the label. In the example, the name of each attribute is followed by colon, the elements of each attribute are separated by commas, and the attributes are separated by semicolons.

Following subsection presents a use case where an appropriate configuration of the metadata in the FDT is essential to provide a good QoE to users.

2.6.3. Shopping center

An interesting use case is content distribution in malls, where clients can receive on their mobile devices through a wireless network (such as Wi-Fi) different information such as the catalogue of a certain shop or discount vouchers.

In these scenarios there are two key premises to take into account:

- On the one hand, the **file distribution network**. Providing multimedia content delivery services for a high number of users (each one with different preferences) gathered in a relatively small area is not an easy task. Current solutions need great investments in order to be capable of satisfying the demand of a large amount of users. In overcrowded public spaces, basically, the bottleneck is the wireless access, which becomes saturated due to the amount of simultaneous connections, thus avoiding users to access available services with an acceptable quality. In these situations, it results very interesting to explode the use of broadcast networks, where contents are sent to all interested users over only one connection.

- On the other hand, the **personalization** of the contents. Not all contents sent by the content server are relevant for all users since each user has different preferences and needs. Thus, content delivery should be personalized for each user according to their user profile in order to guarantee a good Quality of Experience. For example, a runner user in a shopping center may be interested in contents about sport sales.

This use case presents a system for the delivery of personalized multimedia contents adapted to the users preferences and their context. This system allows the content server to broadcast multimedia contents to an unlimited number of users in an efficient manner, by using the FLUTE protocol.

Also, the user context is used to download the most appropriates contents to each user according to their needs and preferences. In this way, in reception, relevant multimedia contents for each user are downloaded automatically, so users do not need to look for contents manually. When accessing to the mall, users register so as to inform the server about the arrival of a new user.

Fig. 2.19 shows the main building blocks that compose the system as well as their interdependences. In the figure, we can see three main blocks: the content server, the content delivery server and the client.

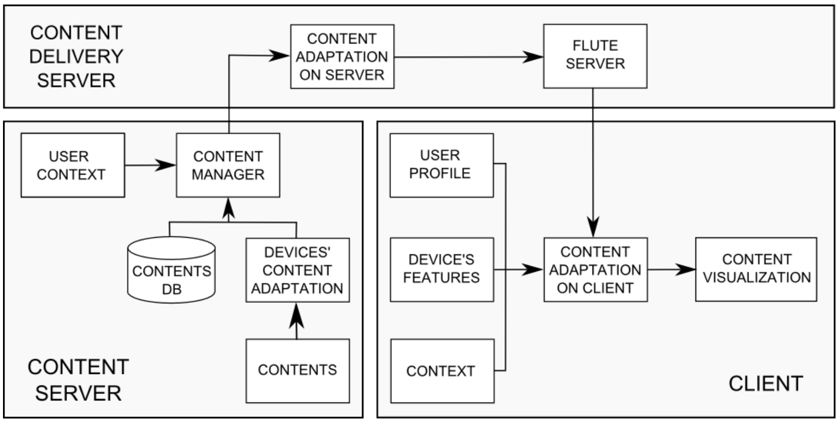


Fig. 2.19. Block diagram of the file distribution system [P.7].

First, the content server manages and provides the different contents and the available metadata. Each content has different metadata associated. For instance, a file can have following metadata: “Fashion”, “Woman”, “Spring”. For each content, this block stores in a database the information needed in order to access both the content and the associated metadata. This information is related to the user context (e.g., location, time, etc.) and it is used by the content adaptation module so as to choose if a certain content is going to be sent or not depending on the users of the system. In this way, the content adaptation module adapts the contents to the different reception devices. Finally, the content manager sends the most relevant information for the user context to the content delivery server.

Then, the content delivery server is in charge of broadcasting the files by means of the FLUTE server. The content adaptation module receives the most appropriate files for the user context from the content server. Contents are sent through sessions and channels using file carousels. These carousels are dynamic, so the files sent and their metadata can change during the transmission. As mentioned, the information about the files sent in each channel and session is sent through the FDT. Also, the FDT includes the metadata of each file, in the way we proposed in Section 2.6.2. On the other hand, the transmission channel is Wi-Fi.

The client block receives all the contents from the FLUTE server and filters this information in order to personalize the contents for each user. In particular, the client application has a module (content adaptation module) that uses the information of the user profile, device features and context information to adapt to the multimedia contents to each user. This adaptation is carried out by means of two mechanisms. On the one hand, a pre-caching is used, downloading useful contents for users without the need of being requested by the application. In this way, the access time to some contents (the most appropriate for each user) is minimized, thus improving the QoE. In order to determine the utility of files for a user and manage the local storage capacity, content filtering techniques are used. On the other hand, the user interface is adapted to the user needs at each moment, minimizing the annoyance and therefore increasing the QoE of the user. For each situation, the system chooses the most appropriate interaction mechanisms for informing the user about new relevant contents downloaded. This adaptation is carried out by the content visualization module automatically.

Finally, it should be noted that an appropriate configuration of both the FLUTE sessions and FLUTE channels can help to personalize the service more efficiently. For instance, it is possible to create as many channels as categories defined (one channel used to send fashion contents, another one for sports…). In this way, user applications only need to connect to those channels which are interesting for the user, thus reducing computational resources (and therefore battery consumption), which it is very important in mobile devices.

A detailed explanation of this use case can be found in a paper submitted for publication [J.4].

2.6.4. Touristic services

This use case, similar to the previous one, is related to the provision of touristic services. Specifically, the multimedia tourism service presented in this use case targets mobile devices equipped with broadband and multicast (or broadcast) access capabilities. The service presents personalized tourism guides to visitors by implementing a personalization engine that adapts the content in the guide to the profile of the visitor.

Tourists access the service through terminals equipped with the appropriate connectivity, i.e. broadband and broadcast access. When users access the service, they receive an electronic tourism guide in their mobile phones, consisted of a list of resources related to tourist locations, events, facilities or amenities in the tourist area. Apart from content assets, the platform operator can also include links to third-party services into the resource descriptions, through the mobile telephony network, such as sending message templates, get contact information, access to external links, make automated reservations, buy tickets, or hire a baby-sitter via web services. Thus, users can browse the list of touristic resources and then access an interactive multimedia description of each resource in the list.

The electronic tourism guide is generated by a recommender engine every time a user access the service. The recommender uses the metadata information about each resource to create a personalized list of items for each user, taking into account the user preferences, their history and the experience of other users with similar profiles. For that purpose, the recommender engine communicates with the users profiles database and with the Content Management System (CMS) to obtain the assets offered by the tourist agency, as Fig. 2.20 depicts.

But the main characteristic of this guide and the service is that the interactive multimedia descriptions of the resources are not delivered together with the tourist guide. Instead, the platform uses broadcast networks to distribute the most popular elements, while the rest are available through the broadband networks, together with the interactive services. When the user selects one of the resources, the client application fetches the associated description from either the broadcast or the broadband network, according to the access information provided by the tourism guide. In this way, the load of the networks is balanced in a transparent way for the user, reducing the congestion and improving the performance of the service. While the user browses the personalized list of resources, the client application joins the broadcast channel in order to cache the resource descriptions transmitted therein. When the user selects a resource it will first check the cache memory and in the event of a cache miss, it will access a web server to fetch the description of that particular service. Fig. 2.20 presents the temporal diagram of the service in a simple manner.

The platform uses IP protocols (namely FLUTE and HTTP) to deliver the data. In this way, the service can be offered in a variety of networks, as long as there is support for IP multicast. This use case is described in [68].

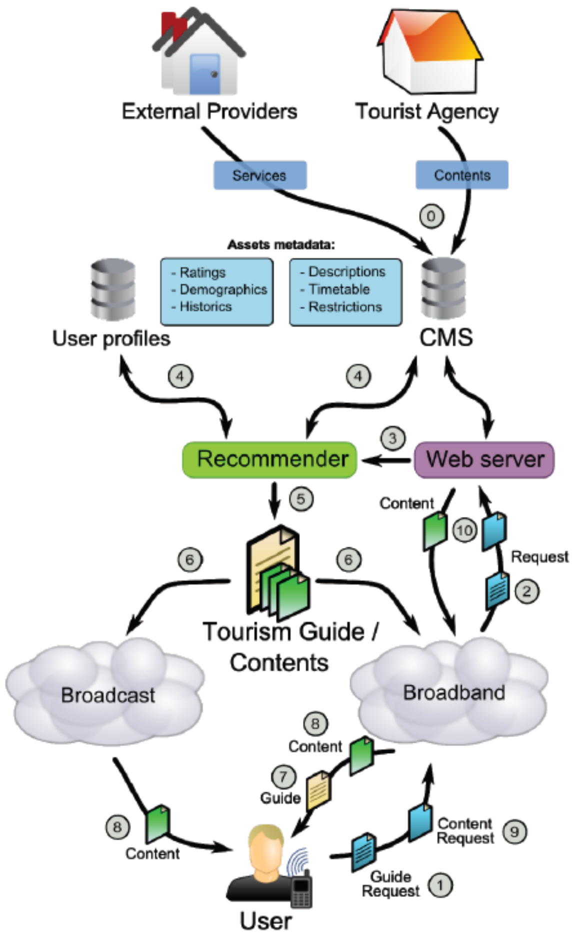


Fig. 2.20. Diagram of the touristic service [P.3].

2.6.5. Background push content download services

This use case proposes the use of background Content Download Services (CDS) to deliver pre-produced television content through existing broadcast networks. The proposal is based on the use of residual capacity in broadcast networks to push popular and pre-produced content to storage capacity in customer premises equipment. This is possible since, due to the variable nature of video streaming, network operators dedicate a considerable amount of network resources to live streaming video, so there is an excess of reserved capacity that can be used to provision other services with no instant capacity requirements, such as unidirectional background push content download services.

As an example, Fig. 2.21 shows the architecture of the unidirectional background CDS proposed in [69]. A CDS delivers content using FLUTE from a content repository, together with metadata of the content sent (Content Descriptions). The Scheduler establishes the order at which files are delivered, according to parameters like their size or popularity. Later, the CDS is delivered over a background virtual channel by inserting packets from the CDS whenever there is capacity available in the network (Opportunistic Insertion). In this way, the CDS is delivered over the background channel, made of the residual transmission capacity in the reservations of the primary service (a television service).

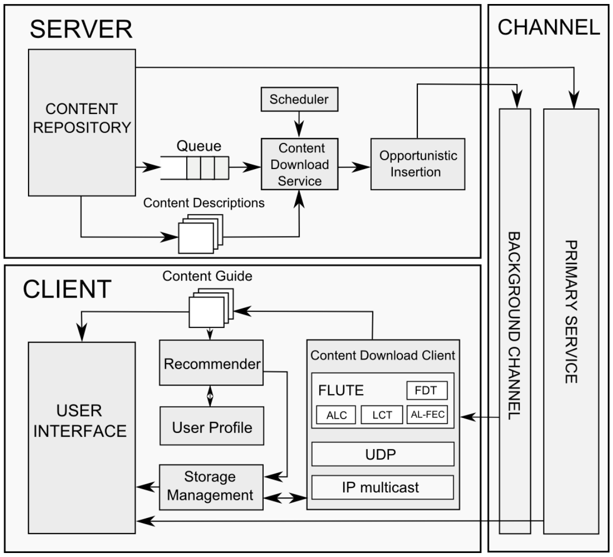


Fig. 2.21. Unidirectional background push CDS architecture [J.2].

Regarding the client side, the CDS client is based on the FLUTE protocol, as figure shows. The storage memory is controlled by the Storage Management, which ensures that the client uses only the storage capacity reserved for the background service. The Recommender uses feedback from user interaction and the metadata in the Content Guide to generate the User Profile. In this way, the Recommender is able to provide the Storage Management with an estimation of the usefulness of each content item. In turn, the Storage Management uses this information to filter the contents offered in the CDS, in order to keep in storage only the items that better fit the user preferences. This introduces some level of personalization to the service, thus improving the Quality of Experience.

A detailed explanation of the architecture proposed in this use case as well as a complete evaluation of the different elements that build the system can be found in [69].

2.6.6. Hybrid FLUTE/DASH for video delivery

Apart from sending files, FLUTE can be useful to send video. Specifically, there are three general ways of consuming video: stored video, pseudo-streaming and live streaming. In this sense, the transmission of video for their posterior visualization corresponds to the case of a file delivery, once the video is completely downloaded the user can consume it. On the other hand, in pseudo-streaming users have to wait a certain time in order to consume the video. This time depends on the transmission rate and the buffer size. Finally, in live streaming users consume the video in real time without waiting.

As we have mentioned in the related work, FLUTE and ALC can be used to send pseudo-streaming services. In this sense, this use case proposes the use of a hybrid FLUTE/DASH (Dynamic Adaptive Streaming over HTTP) to provide mobile video streaming services over broadcast wireless networks. This approach results rather innovative comparing it to current related work. Thus, it is worth highlighting the standards MBMS [33] and eMBMS [34] of the 3GPP Project, which propose the use of RTP (Real Time Transport Protocol) for video streaming and FLUTE to download files. Also, FLUTE is proposed to send information related to DASH (such as signaling or DASH segments). Unlike this use case, both standards do not propose FLUTE to send video nor DASH as a video repair service. A related reference is [70], which presents an overview of the challenges of mobile video streaming, such as DASH over eMBMS. In fact, this work explains the usage of FLUTE for transmitting DASH segments. In this sense, the use of the same segmentation scheme in both protocols can yield innovative ways to distribute video segments in wireless networks with broadcast or multicast support.

DASH [71] is a new ISO (International Organization for Standardization) standard for the transmission of on-demand and live content with time-shifting capabilities. DASH is based on multimedia file segmentation. Each multimedia file is encoded in different qualities and every quality file is split into small portions called segments. In order to access the whole multimedia content, clients select which quality of each segment they want to download. DASH defines a manifest file, called Media Presentation Description (MPD), which describes the multimedia content, the different qualities and how the content is split into segments. Each video service is represented by a media presentation, which is a collection of time dependent media items, as shown at the bottom of Fig. 2.22. Media presentations are composed by a sequence of periods. Periods are time intervals along the duration of the media, which cannot overlap. Each period has different encoded alternatives, referred to as representations.

The scenario regarded in this use case is depicted in Fig. 2.22. There are many interested clients inside the service area and the delivery of the video through unicast connections can cause congestion in the wireless link. In order to avoid this, video services use the hybrid broadcast/unicast streaming technology proposed in this use case.

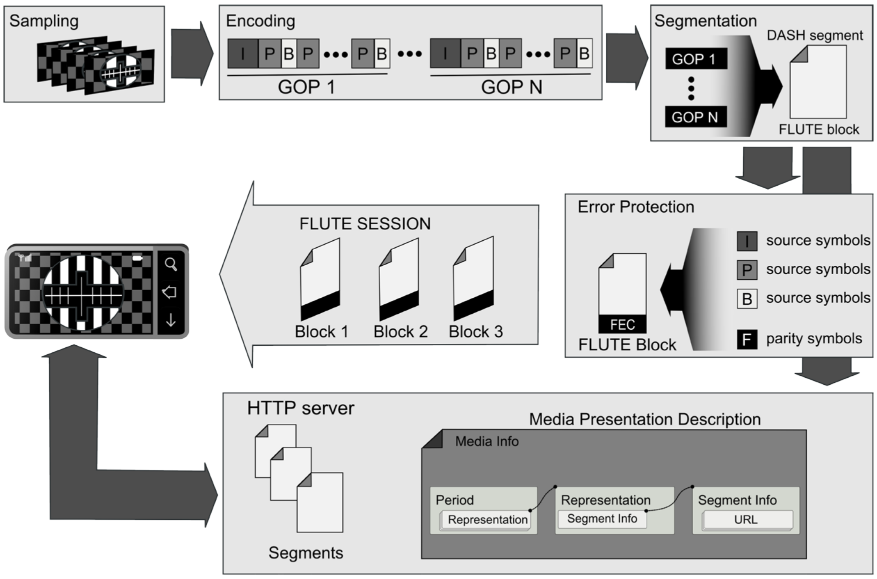


Fig. 2.22. Hybrid FLUTE/DASH video delivery architecture [J.3].

Regarding the sampling and encoding processes, modern video encoders based on the MPEG-4 family of video coding standards, such as H.264/AVC, exploit adjacent frames and nearby pixel correlation to reduce temporal (inter-frame) and spatial (intra-frame) redundancy as well as perceptually unimportant information. Thus, video frames are classified into I, P and B-frames, depending on the coding dependency. Following this scheme, a video stream is composed of fully decodable units called Group of Pictures (GoPs). Within a GoP, I-frames are encoded independently of any other frame in that GoP. Alternatively, P-frames use motion/estimation compensation based on information related to a previous frame. Finally, B-frames can reference previous and subsequent frames. Remark that this is a basic coding scheme, whereas state-of-the-art encoders support more advanced encoding combinations and techniques [72]. According to this basic coding scheme, if an error is produced during transmission in any I-frame, this error is propagated throughout the GoP due to inter-frame dependencies. However, errors in P or B-frames only affect dependent frames, causing lesser video distortions.

After the encoding process the video is segmented according to a given segmentation policy, generating DASH segments and FLUTE blocks. The information related to the DASH segmentation is indicated in the MPD. Then, clients can access the segments through a FLUTE session and a DASH server. The flexibility regarding the segmentation of a video in FLUTE and DASH allows to combine both technologies to download video frames. It is worth noting that in FLUTE the encoding process is carried out in each block and thus, different blocks can use different AL-FEC codes. This allows to provide different protection to different blocks, so it is possible to use

UEP techniques in a simple way. Note that UEP has been proposed in several works regarding video streaming, such as [73].

In the proposal, presented in [J.3], each FLUTE block represents a video frame, and is formed by *n* encoding symbols: *k* source symbols and *n* - *k* parity symbols. In this use case, the AL-FEC block applies different AL-FEC code rates to source symbols belonging to I, P or B-frames, that is, Unequal Error Protection. I-frames have more protection (and therefore more parity symbols) than P and B-frames, since I-frames are more important, as discussed above. Also, blocks of different frames can have different number of symbols, depending on the size of the frame. It is worth remembering that the coding parameters regarding each block are included in a FLUTE extension header, in a format that depends on the type of coding used.

Clients connect to the FLUTE multicast session (before the server starts the transmission) and filter the packets belonging to the objects they want to download. Thus, clients receive the different symbols that compose each block and rebuild the corresponding GoPs. The parity symbols received are used to recover the source symbols lost. If after applying AL-FEC decoding some packets have not been recovered, clients are able to request only the missing data to the DASH server through a unicast connection. As mentioned, clients discover the information referred to the DASH server by means of the manifest file (MPD), which identifies the alternative locations for each segment in their respective segment information descriptions.

The way DASH carries out the segmentation is very flexible, and each segment could contain from only one video frame to several GoPs. In [J.3] it is considered that there is a DASH segment per GoP, and each GoP is further divided into sub-segments. In any case, in the event of losses, a client can request to the DASH server either the entire GoP, or only the bytes lost.

Clients start playing the video when a certain number of GoPs are received, thus providing low initial start-up latency. This number depends on the buffer of the client. In order to provide a continuous playback of the video, the transmission bandwidth has to be higher than or equal to the playback rate of the video. When clients are displaying the content of GoP *n*, they are receiving packets of following GoPs through the hybrid FLUTE/DASH network. On the other hand, the FLUTE transmission rate and the number of requests to the DASH server depend on the bandwidth available to fulfill with the playback rate condition.

A complete evaluation of this use case is presented in [J.3]. As a general conclusion, the proposed hybrid broadcast/unicast architecture based on FLUTE and DASH can help to reduce considerably the bandwidth without degrading neither the video quality nor the start-up latency, thanks to the use of AL-FEC mechanisms to repair data (especially when using UEP techniques) and DASH to recover lost frames.

2.7. Conclusions

This chapter has presented the state of the art related to the most important technologies used within this thesis. Specifically, this chapter has explained in detail the FLUTE protocol as well as LDPC codes, presenting the related work.

Furthermore, this chapter has also presented an implementation of a file transmission server/client based on the FLUTE protocol, which includes an LDPC library for encoding/decoding purposes. This implementation has been developed in the context of this thesis in order to evaluate the several studies presented in this work.

On the other hand, the different use cases explained have been useful to know some of the multiple applications of multimedia file distribution in multicast networks. Among them, we have seen how multicast file delivery is useful in crowded environments such as sports events or tourist sites. Also, we have explained background push content download services and we have presented a hybrid FLUTE/DASH system to provide video delivery services.

In this sense, the use of personalization mechanisms allows to improve the Quality of Experience of users in multicast file delivery. Also, in all use cases presented, one of the key elements is to offer a reliable service, minimizing the effect of errors in the transmission. In this sense, following chapters of this thesis work analyze in depth the use of AL-FEC mechanisms based on LDPC codes.

Chapter 3

Evaluation of LDPC Staircase and LDPC Triangle codes

FEC mechanisms improve the reliability of IP content transmissions through the recovery of packets lost. Opposite to ARQ, FEC mechanisms are especially suited to unidirectional environments or to multicast environments where multiple receivers perceive different channel losses, thus making difficult the implementation of mechanisms based on feedback information. Among the different types of FEC codes, this chapter presents a thorough performance evaluation of LDPC codes, based on an implementation developed in the context of this thesis, according to the specifications defined by RFC 5170 for the usage of LDPC codes by push content applications based on the FLUTE protocol. LDPC codes provide a good trade-off between performance and complexity, hence, they are appropriate for mobile applications. Contributions of this chapter include tests conducted with commercial mobile phones connected to the push content download server over a Wi-Fi network. The evaluation highlights the advantages of using packet level FEC encoding in file transmission over unidirectional networks and provides with a comparison between two kinds of LDPC structures: Staircase and Triangle. This is accomplished by calculating the inefficiency ratio as well as the download time of a file using these LDPC structures in different environments. Results show that the implemented LDPC codes can provide inefficiency ratios close to one when the different coding parameters (as the code rate or the number of blocks) are configured to an optimal value that depends on the packet loss rate. Also, results reflect how an appropriate configuration of the coding parameters can reduce the download time.

3.1. Introduction

This chapter evaluates LDPC Staircase and LDPC Triangle codes, which were explained in Chapter 2. Apart from comparing both LDPC structures, the results also present a comparison of these structures regarding Compact No-Code, that is, the coding where no FEC is applied (No-FEC).

Section 3.2 explains the methodology used to carry out the evaluation. The evaluation parameters are presented as well as the different studies (detailing their configuration parameters). The performance of the implementation of LDPC codes has been assessed through several tests, which are presented in Section 3.3. Finally, Section 3.4 presents the main conclusions of this chapter.

3.2. Evaluation methodology

In the tests carried out, two different scenarios have been proposed, as Fig. 3.1 shows. In the first scenario the server and the client are in the same machine to avoid uncontrolled packet loss in the network. In order to simulate packet losses in the channel, a two state Markov model has been implemented in the FLUTE client.

In the second scenario, the FLUTE client is a mobile phone and connects to the server through a Wi-Fi channel. In this sense, one of the main contributions of this chapter is the evaluation of LDPC codes with mobile devices through wireless networks.

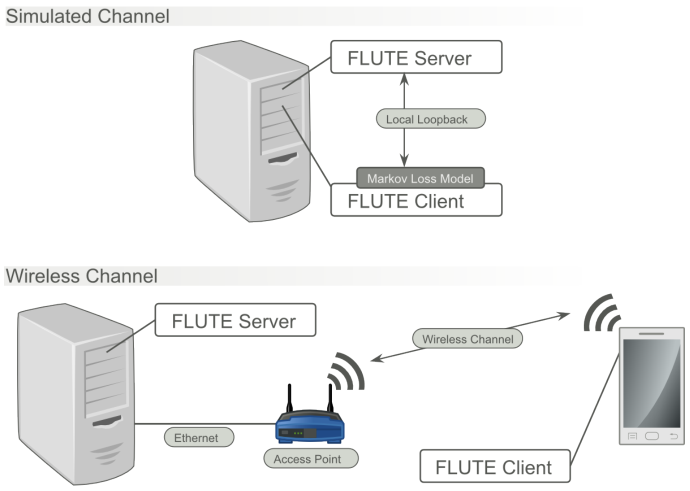


Fig. 3.1. Testbed.

In order to see the losses detected in the wireless channel in our scenario, a study over the Wi-Fi multicast losses is presented. The study has been carried out in a typical laboratory indoor environment, in which there are several computers and access points. The measurements assessed the number of packets per cycle received by the mobile terminal, so it is possible to calculate the percentage of lost packets. Fig. 3.2 depicts the results of a study made between 9.30 am and 1.00 pm.

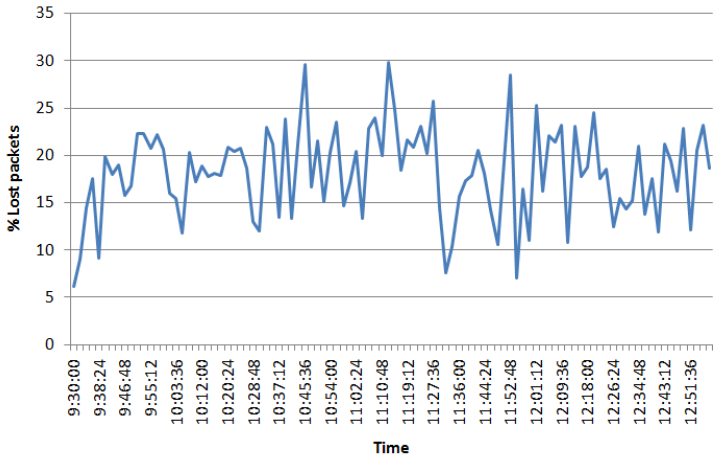


Fig. 3.2. Evaluation of losses in Wi-Fi in laboratory environment.

As figure shows, the percentage of losses is time-dependent. In general, the percentage of losses in our trial environment is between 15% and 25%. In order to obtain accurate measurements, the tests carried out (which are presented in the next sections) have been done in different days and hours, using different transmission rates.

On the other hand, the parameters evaluated in this chapter are:

- **Inefficiency ratio**: represents the relation between the number of received packets needed to decode a file and the number of source packets that make up the file. The less the inefficiency ratio the more efficient is the coding. Ideally this value is 1.
- **Download time**: time passed since the client receives the first packet of the file to download until the file is completely downloaded.
- **Number of carousel cycles** needed to rebuild the file to download. In the studies carried out we use carousels to send files. In this way, clients can complete their downloads if they have suffered packet losses in previous transmissions of the file.

Table 3.1 shows the coding parameters used in each study (presented in Sections 3.3.1-3.3.6), emphasizing in bold and italics the parameters evaluated in each case. In the table, t_D refers to the download time. The file size is expressed in packets with a payload size of 1428 bytes. We have used this value according to the results presented in [45], and taking into account the maximum transfer unit (MTU) of Ethernet (1500 bytes). On the other hand, the transmission rate used is 5 Mb/s, although the conclusions obtained are independent on the transmission rate. Finally, the number of measurements accomplishes good 99% confidence intervals in all scenarios.

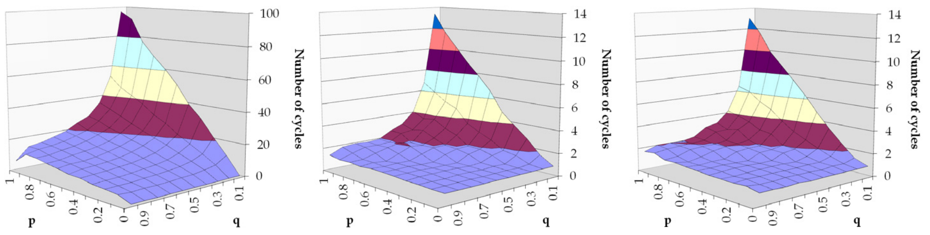
Table 3.1. Study parameters.

Study	**3.3.1**	**3.3.2**	**3.3.3**	**3.3.4**	**3.3.5**	**3.3.6**
Evaluation parameter	n_{cycles}, t_D	Inef. ratio, t_D	Inef. ratio, t_D	Inef. ratio, t_D	Inef. ratio, t_D	Inef. ratio, t_D
Tx. Model	Sequential	***Sequential, Random***	Random	Random	Random	Random
Code rate	2/3	2/3	***[0.2, 0.9]***	2/3	2/3	2/3
File size (packets)	1500	1500	1500	***[10, 10000]***	1500	1500
Blocks	1	1	1	1	***[1, 150]***	1
N1	3	3	3	3	3	***[3, 8]***
Channel	Simulated	Simulated	Simulated, Wireless	Simulated, Wireless	Simulated, Wireless	Simulated, Wireless

3.3. Results and analysis

3.3.1. Number of rebuilding cycles and download time

The first study shows the number of cycles that one client needs to rebuild a file as well as the download time based on the channel losses, which are simulated with a two state Markov model. It should be remembered that, in this model, *p* indicates the probability that a packet is lost when the previous was received, and *q* indicates the probability that a packet is received when the previous was lost. A brief explanation of the Markov Model can be found in Chapter 4 (Section 4.3.2). The graphs of Fig. 3.3 show the results obtained regarding the number of cycles.



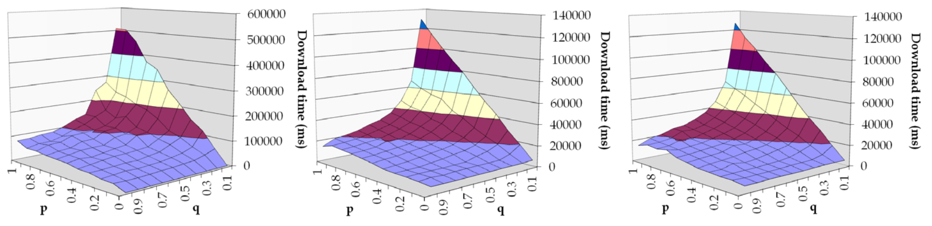
(a) No-FEC (b) LDPC Staircase (c) LDPC Triangle

Fig. 3.3. Number of cycles depending on coding.

Noting the scale of each graph, we can clearly see the convenience of using coding (in LDPC, 15 cycles are not exceeded, whereas in No-FEC it arrives until almost 100 cycles with high losses). The tendency is the same in the three codes, but the difference between them is higher when the losses increase. In low-loss environments (that is, when *p* is low and *q* is high), the graphs show that LDPC codes (both Staircase and

Triangle) present a more stable behavior and close to 1, whereas when no coding is used the number of cycles grows fast with a slight increase of the losses.

Obviously, the results of the download time offer the same conclusions, since the download time is directly related to the number of cycles needed to rebuild the file, as Fig. 3.4 shows (note again the different scale of No-FEC).



(a) No-FEC (b) LDPC Staircase (c) LDPC Triangle

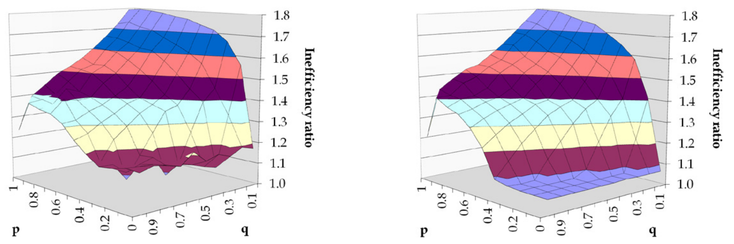
Fig. 3.4. Download time depending on coding.

Comparing both LDPC structures, we can see that the results regarding the number of cycles and the download time are very similar. In following studies we will analyze in further detail the differences of these two structures.

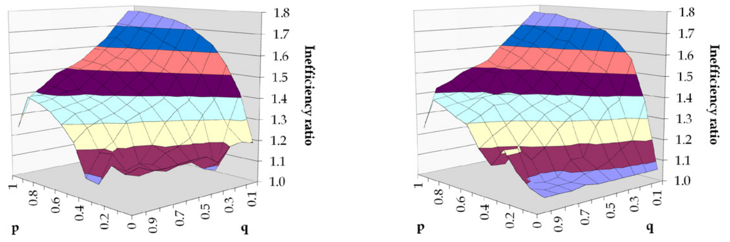
3.3.2. Transmission model

This study shows how the transmission model affects the coding efficiency. To that extent, two models are analyzed: a sequential model, in which packets are sent in order (first source symbols and then parity symbols); and a random model, where packets are transmitted randomly (interleaving source and parity symbols). First, the inefficiency ratio is analyzed. The results are depicted in the graphs of Fig. 3.5.

The figures show that in typical lossy environments (low p and high q), the random transmission model has a better behavior and is more efficient than the sequential one. That is logical if we consider that, in wireless channels, losses are usually produced in bursts and, as in LDPC codes a parity symbol depends on the previous symbol, the loss of consecutive packets prevents the rebuild of the source symbol. With high losses the behavior of both models is similar.



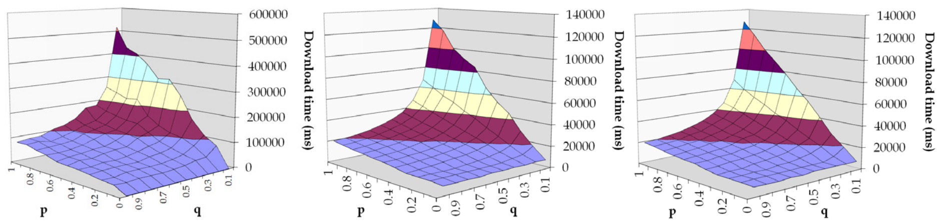
(a) LDPC Staircase, Sequential (b) LDPC Staircase, Random



(c) LDPC Triangle, Sequential (d) LDPC Triangle, Random

Fig. 3.5. Transmission model evaluation (inefficiency ratio).

On the other hand, the evaluation of the download time using the random transmission model is shown in Fig. 3.6:



(a) No-FEC (b) LDPC Staircase (c) LDPC Triangle

Fig. 3.6. Download time depending on coding (random model).

Comparing this figure with the results presented in Fig. 3.4, we can see that both transmission models offer similar download times, overall when losses are high.

According to these conclusions, the evaluations presented throughout this thesis dissertation will consider the random transmission model.

3.3.3. Code rate

As mentioned, the code rate is a basic parameter of push content download services. It is defined as k/n, that is, it represents the relation between the number of source symbols and the number of encoding symbols of a file. The number of parity symbols is, hence, $n - k$. Another parameter used is the FEC ratio, defined as n/k, which is the inverse of the code rate.

Fig. 3.7 shows how the code rate affects the inefficiency ratio in a lossless channel (the code rate axis has been expanded in order to see in detail the behavior of each structure).

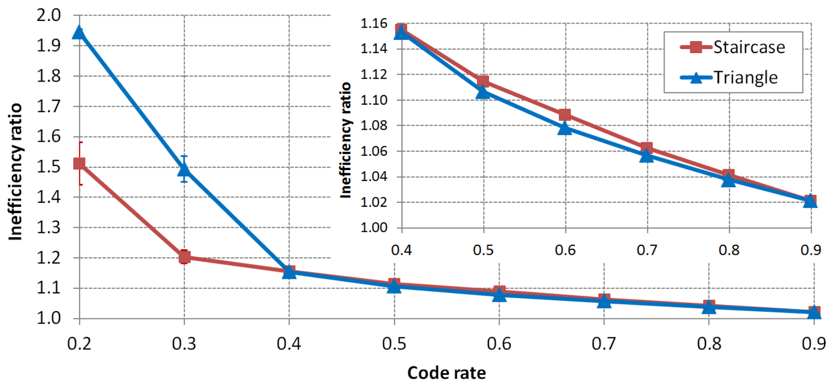


Fig. 3.7. Code rate evaluation in a lossless channel (inefficiency ratio).

The higher the code rate the lower (and the better) the inefficiency ratio. Figure shows that LDPC Staircase structure is more efficient when the code rate is lower than 0.4, whereas LDPC Triangle provides better results for code rates higher than this value. Although for values of code rate larger than 0.4 the difference between both structures appears to be small, it could be very meaningful when big files are sent.

As the code rate is higher, less parity packets are sent, so in lossless environments the inefficiency ratio will be lower (since less "useless" packets are received). Ideally, in a lossless channel, if the code rate is 1 (that is, no coding is used) the inefficiency ratio is 1. But, unfortunately, most of channels have losses. Before seeing the analysis in a wireless environment, we study the behavior in a loss environment, modeling losses with the two state Markov model. Fig. 3.8 shows the evaluation of the inefficiency ratio of LDPC codes in an emulated channel with parameters p = 0.1 and q = 0.3, which represents a packet loss rate of 25% according to Chapter 4 –equation (4.7).

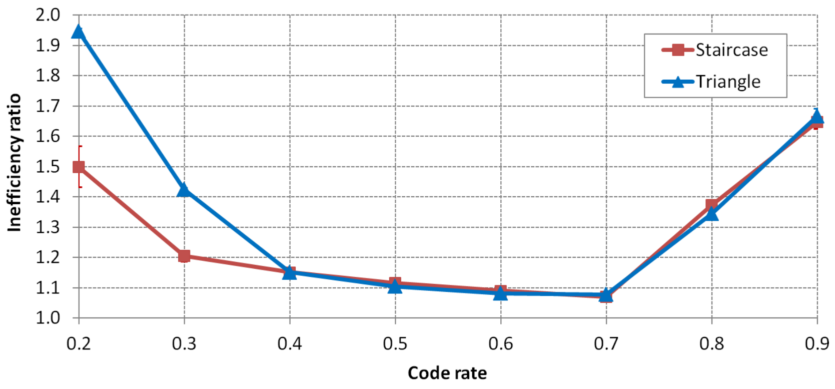


Fig. 3.8. Code rate evaluation in a loss channel (inefficiency ratio), p=0.1, q=0.3.

We conclude that for choosing an appropriate code rate it is necessary to bear in mind the losses of the channel. Using high code rates could cause that the information is not protected appropriately, hence increasing the inefficiency ratio. For instance, for the channel evaluated in Fig. 3.8, the best code rate is 0.7. In this sense, we arrive to the same conclusion when evaluating the download time in the same losses environment (Fig. 3.9). In Chapter 4 we will analyze in detail that there is an optimum code rate that minimizes the download time for each percentage of losses.

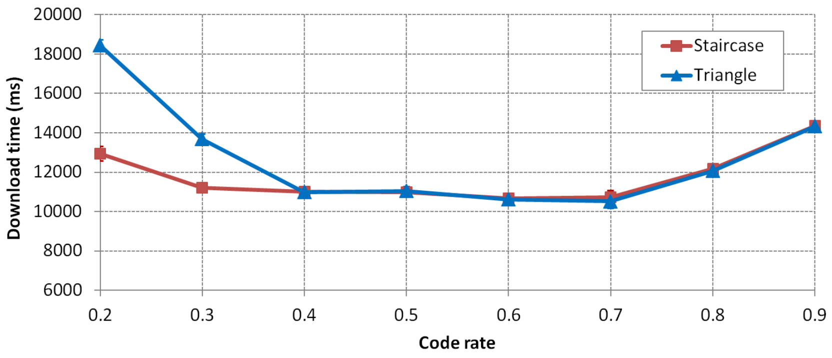


Fig. 3.9. Code rate evaluation in a loss channel (download time), p=0.1, q=0.3.

Moreover, the behavior of the code rate has been tested in a Wi-Fi environment with mobile devices. Fig. 3.10 gathers the results of this study, where the conclusions reached in the previous studies still hold. LDPC Staircase is more efficient with code rates lower than 0.4, whereas for code rates higher than 0.4 the behavior of both LDPC structures is similar. Depending on the channel, there are code rates that minimize the inefficiency ratio. The values of the inefficiency ratio are rather higher than in the Fig. 3.7, due to the losses of the channel.

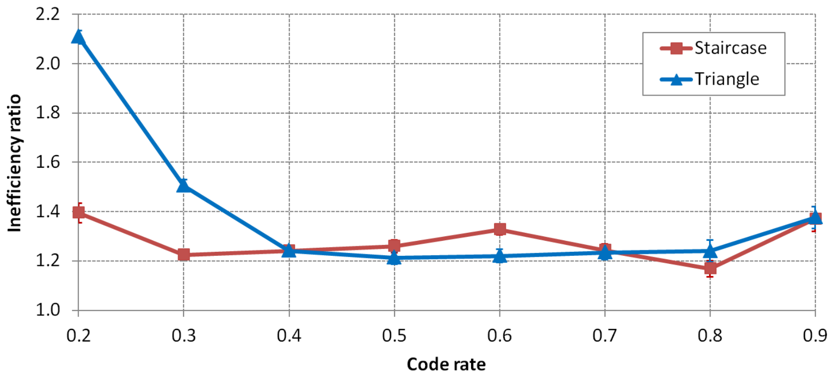


Fig. 3.10. Code rate evaluation with a mobile device in Wi-Fi environment.

3.3.4. File size

As we have seen, using any coding mechanism makes the transmission more efficient. This improvement depends on the size of the information sent. Fig. 3.11 shows a comparison among LDPC (an average of LDPC Staircase and LDPC Triangle codes) and No-FEC regarding the inefficiency ratio for different file sizes. The study has been carried out in a Wi-Fi channel.

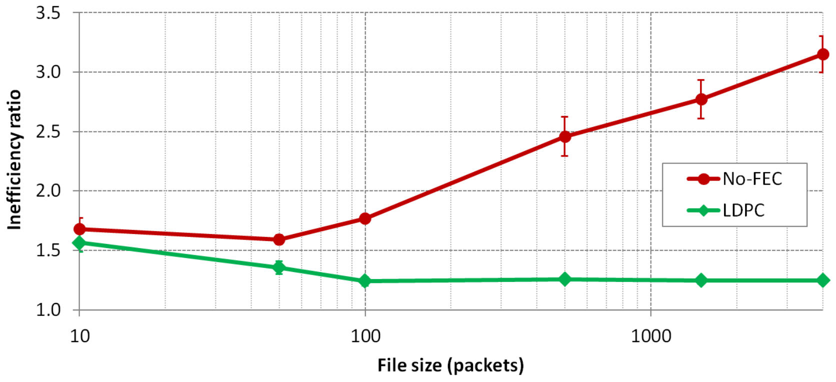


Fig. 3.11. Comparison between No-FEC and LDPC depending on file size in a Wi-Fi channel.

The behavior of the two coding mechanisms is completely different. With No-FEC the larger the file size, the higher (and worse) the inefficiency ratio, whereas with LDPC the opposite holds. The advantages of using FEC coding are more evident when large files are sent.

A deeper study of LDPC depending on file size is explained next. First, using a channel with no losses. The results are shown in Fig. 3.12.

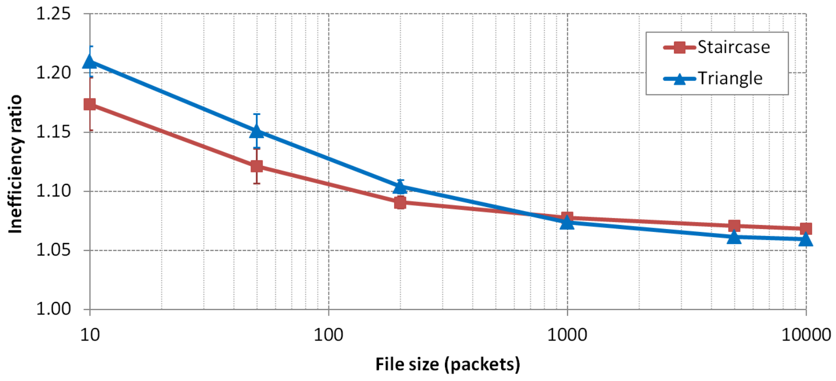


Fig. 3.12. File size evaluation in a lossless channel (inefficiency ratio).

LDPC codes are more efficient when large files are sent, as the graph shows. For instance, with files of 10 000 packets size (over 14 MB), for the Triangle structure the inefficiency ratio is 1.0593. This means that it is only needed to receive a 5.93% more of the packets which make up a file to rebuild it. That is, reliability is being provided to the communication but without increasing the rebuild time in reception excessively.

With regard to the LDPC structure, the figure shows that both structures have a similar behavior although, in general, Staircase offers better results than Triangle with small files, whereas with large files LDPC Triangle has a better inefficiency ratio.

The study in a wireless environment reflects the same behavior of both structures, as Fig. 3.13 shows.

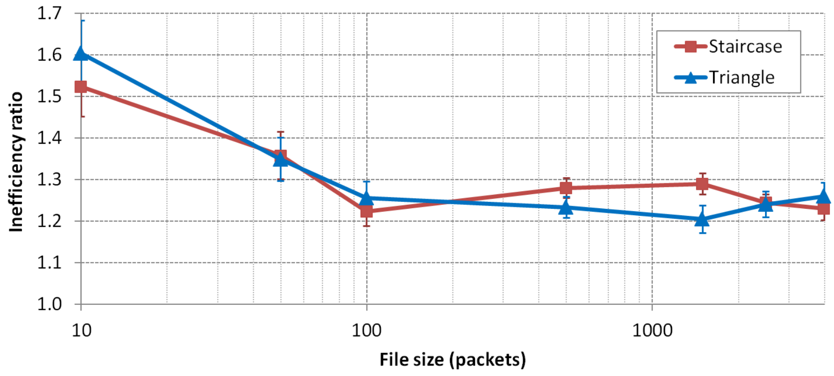


Fig. 3.13. File size evaluation with a mobile device in a Wi-Fi channel (inefficiency ratio).

It should be highlighted that the conclusions reached regarding the file size and the code rate are consistent with those found in [51].

Regarding the download time, as expected, the download time increases linearly with the size of the file. Once again, LDPC Staircase and LDPC Triangle provide similar download times for all file sizes.

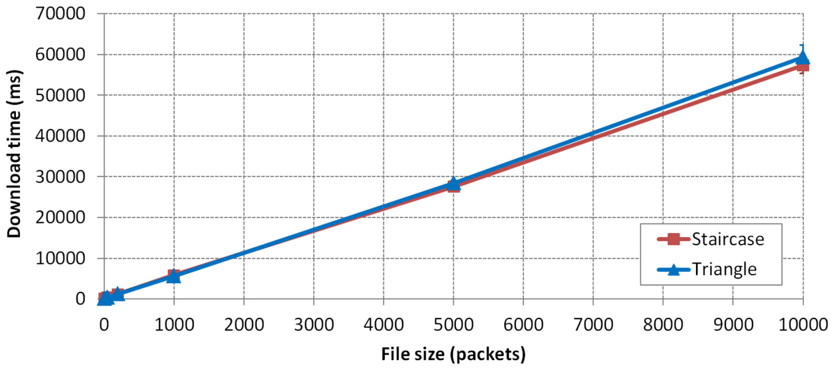


Fig. 3.14. File size evaluation in a lossless channel (download time).

3.3.5. Number of blocks

A related study is the number of blocks in which a file is divided. In this sense, Fig. 3.15 shows the inefficiency ratio measured when the number of blocks changes.

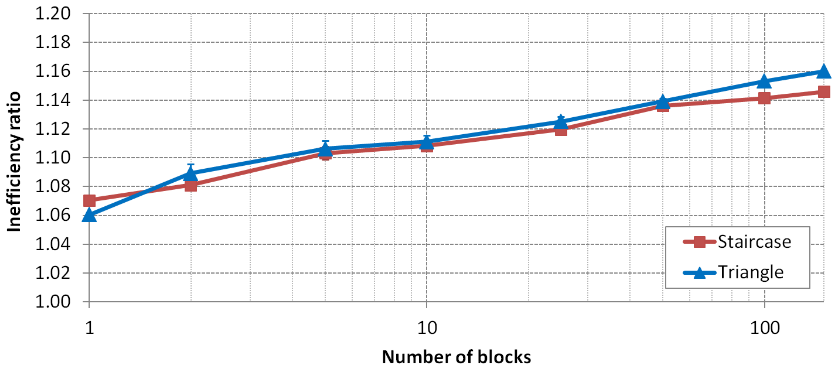


Fig. 3.15. Number of blocks evaluation in a lossless channel (inefficiency ratio).

The inefficiency ratio increases with the number of blocks used and therefore, in terms of efficiency, it is better to use one block in the delivery of files. That is logical considering that, if a high number of blocks is used, each block has fewer packets and, as we have seen before, LDPC codes are less efficient with small files. Nevertheless, it could be convenient to use more than one block in order to reduce the memory consumption.

Fig. 3.16 shows the behavior in a mobile device using a Wi-Fi channel.

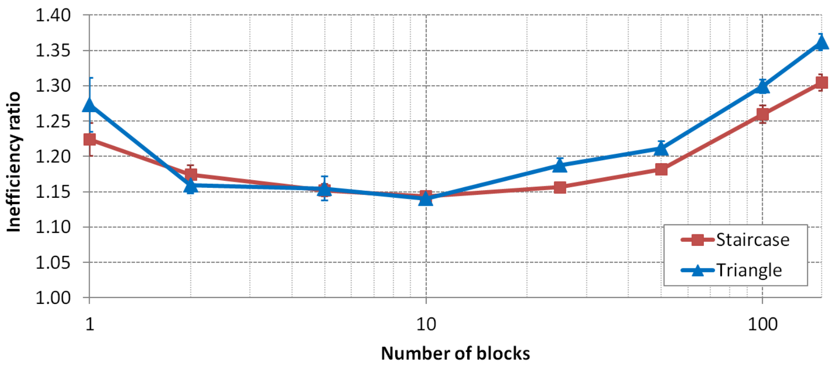


Fig. 3.16. Number of blocks evaluation with a mobile in a Wi-Fi channel (inefficiency ratio).

The results are very similar to those shown in Fig. 3.15, except for the value of 1 block. The tendency is the same: the higher the number of blocks, the higher the inefficiency ratio. The LDPC Staircase structure has a better behavior than Triangle when the number of blocks increases.

The download time is also affected by the number of blocks: the less the number of blocks the lower the download time, as Fig. 3.17 shows for a lossless channel:

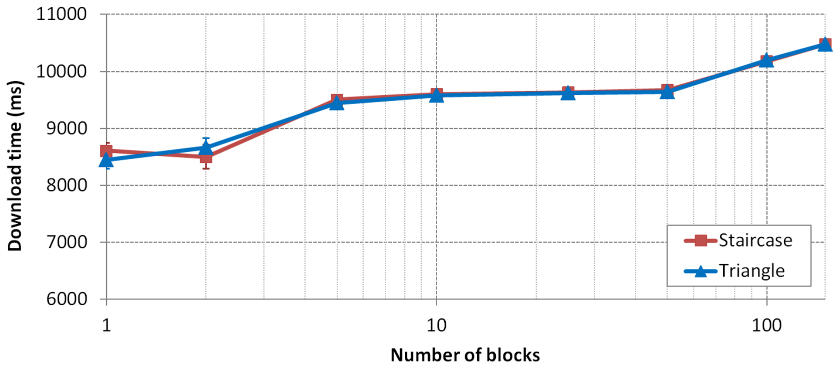


Fig. 3.17. Number of blocks evaluation with a mobile in a lossless channel (download time).

3.3.6. Number of 1s in the parity check matrix

In the parity matrix creation (specifically in the left submatrix), each source symbol can be part of a certain number of equations (*N1*). This number is fixed for each matrix and it is usually equal to 3, as the RFC 5170 [6] recommends. Fig. 3.18 shows the inefficiency ratio obtained when *N1* varies between 3 and 8 (values under 3 are not allowed), in an evaluation through a lossless channel.

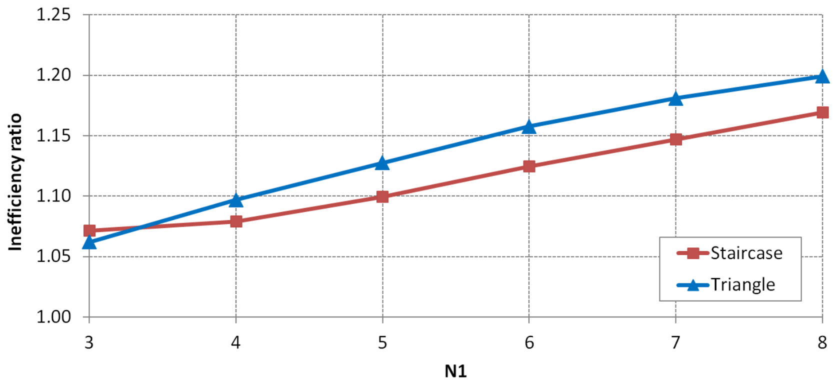


Fig. 3.18. *N1* evaluation in a lossless channel (inefficiency ratio).

Similar results are obtained when the same study is done in a wireless environment, as Fig. 3.19 presents. Both figures show that the inefficiency ratio increases when the parameter *N1* is higher. Moreover, the Staircase structure results more efficient than the Triangle one when *N1* grows. Additionally, the download time evaluation also proves the convenience of using a low value of *N1*, as Fig. 3.20 depicts.

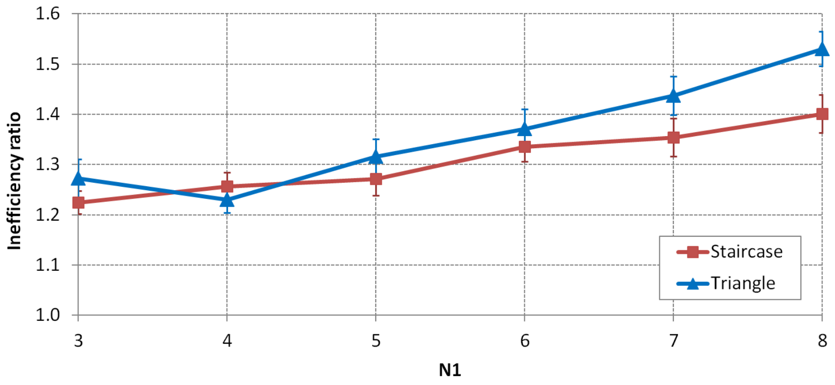


Fig. 3.19. *N1* evaluation with a mobile device in a wireless channel (inefficiency ratio).

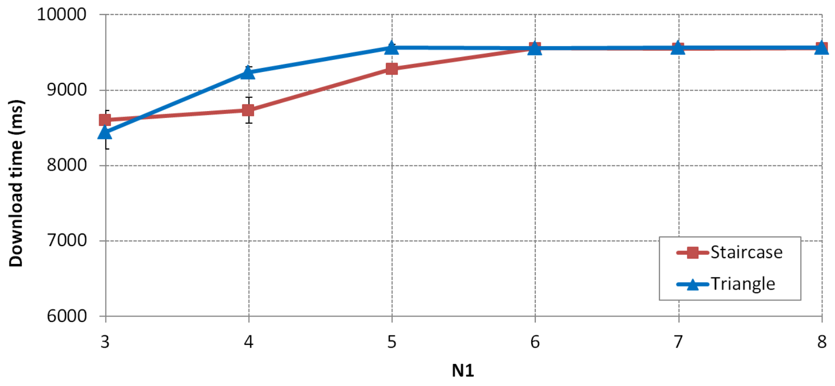


Fig. 3.20. *N1* evaluation with a mobile device in a lossless channel (download time).

Nevertheless, the results depend on the decoding algorithm used. In our case, we have used a simple iterative decoding algorithm (explained in Section 2.5.3) due to its simplicity and its low memory consumption. As we presented in the related work (Section 2.4), there are several studies (such as [52] and [53]) that show that using another decoding algorithms (for instance the ones based on Gaussian elimination scheme) allows to reduce the inefficiency ratio. Those studies reflect that, using a Gaussian elimination scheme, the increase of *N1* means a lower inefficiency ratio for LDPC codes, at the expense of increasing the memory consumption.

Therefore, we conclude this study saying that, using a simple iterative decoding algorithm, an increase of *N1* does not mean an improvement in the inefficiency ratio nor in the download time, so the optimal value is *N1* = 3.

3.4. Conclusions

LDPC codes allow to reduce considerably the number of cycles needed to reconstruct a file and, therefore, the download time. This reduction is bigger in channels with high losses.

On the other hand, the packet delivery scheduling is a parameter that affects the efficiency of the content push download service. In environments with low losses, a random delivery model is more efficient than the sequential one, since it is more resilient to burst packet losses.

LDPC is more efficient with large files and when only one transmission block is used. Regarding LDPC Staircase and LDPC Triangle, both structures offer similar values of inefficiency ratio and download time for different configurations although, in general, LDPC Staircase is more efficient with code rates lower than 0.4 and when short files are sent. In the experiments made with mobile devices in a Wi-Fi network, although the results of the inefficiency ratio and the download time are worse, the conclusions that we have reached are the same. Anyway, the difference between these two structures is pretty minimal.

On the other hand, one of the parameters that affects the memory consumption is the decoding algorithm. The use of other algorithms, such as the Gaussian elimination scheme, improves the inefficiency ratio but increases the required memory by the terminal.

Finally, the optimal coding parameters in each case (code rate, number of blocks…) depend on the transmission characteristics: channel losses, files sent or processing capacities of the receivers. In this sense, the following chapter presents how the performance of LDPC codes can be improved by using an adaptive mechanism according to the losses perceived by the clients.

As a result of the work presented in this chapter, we have published a paper in a national conference [C.6] and in an international journal [J.5].

Chapter 4

Adaptive LDPC AL-FEC codes

This chapter proposes the use of adaptive LDPC AL-FEC codes for content download services over erasure channels. In adaptive LDPC codes, clients inform the content download server of the losses they are experiencing. Using this information, the server makes FEC parity symbols available to the client at an optimum code rate. This chapter presents an analytical model of the proposed adaptive LDPC codes. The model is validated through measurements performed with an application prototype. Additionally, results show the performance of these codes in different scenarios, compared to the performance of non-adaptive AL-FEC, optimum LDPC AL-FEC codes and an ideal rateless code. Adaptive LDPC AL-FEC codes achieve download times similar to ideal rateless codes with less coding complexity, at the expense of an interaction channel between server and clients.

4.1. Introduction

In this chapter, the code rate of an LDPC AL-FEC code is adapted to the erasure rate of the channel as perceived by a particular user. During the transfer process, clients report on the erasure rate they perceived. If needed, the server generates additional FEC rate and inserts it in a multicast channel, so that receivers are notified about the availability of additional FEC parity data and start processing it. The performance of these proposed adaptive LDPC AL-FEC codes is compared to rateless codes, non-adaptive LDPC AL-FEC codes and optimum LDPC AL-FEC codes. The optimum LDPC AL-FEC codes are an ideal implementation of adaptive LDPC codes where the feedback received by the clients is instantaneous.

The rest of the chapter is structured as follows. Section 4.2 presents the test case scenario that sets the basis for the measurements. Section 4.3 analyzes mathematically adaptive LDPC codes, whereas Section 4.4 describes the methodology used for the measurements. Section 4.5 includes the theoretical and the experimental results and their corresponding analysis. Finally, the last section includes some final conclusions about the study.

4.2. System overview

The system proposed uses a hybrid unicast/multicast content delivery mechanism to provide content to users within the boundaries of the service area. It is assumed that the overall capacity of the wireless access is shared between unicast and multicast connections and that there is a limited bandwidth for multicast connections. It is also assumed that users experience a slowly varying channel. Moreover, it is assumed that files in the carousel may change with time. This system may be provided on top of any wireless network technology with multicast support, such as Wi-Fi.

During the delivery process, clients and server use a reporting mechanism through which the server obtains an estimation of the erasure rate perceived by every user. If required, the server generates AL-FEC parity and inserts it in the wireless media. All parity symbols belonging to a code rate are inserted on a separate FLUTE channel. Therefore, every client receives the base ALC layer, with encoding symbols belonging to the base FEC rate and, after some time, they also subscribe to a second ALC channel in which they receive additional AL-FEC parity at a rate adapted to the erasure rate that the user experiences.

Obviously, all multicast channels share the overall maximum bitrate allocated for multicast in the wireless access. Also, multicast and unicast traffic compete for network resources. For this reason, there is no multicast traffic until there are a sufficient number of requests for a content item. Similarly, there are no AL-FEC parity channels until users need it. In order to upper bound the maximum number of channels, all users that experienced similar losses are prompted to join the same multicast channel for additional parity data. By separating the parity packets in different channels according to their encoding rate, receivers only need to process AL-FEC packets at an optimal rate for their channel losses. This multicast scheme achieves lower resource consumption in clients, which is appropriate for mobile devices.

Fig. 4.1 shows a general overview of the proposed scenario.

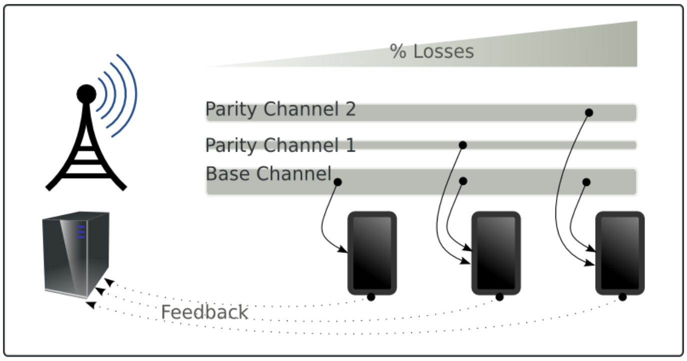


Fig. 4.1. System overview for adaptive AL-FEC codes.

4.3. Theoretical analysis

This section analyzes mathematically adaptive LDPC codes. The main goal is to calculate analytically the download time for multicast FLUTE file transfer services using adaptive LDPC codes. The download time is defined as the time elapsed from a download request until the file is completely downloaded to storage memory. The last encoding symbol received establishes this download time. In this study, each FLUTE packet contains exactly one encoding symbol.

Regarding the transmission, the server sends files cyclically in a file carousel. In order to calculate the download time, it is necessary to know the minimal number of times a carousel is sent, that is, the minimal number of cycles or loops needed by a client to download a file. This number will in turn depend on the packet losses of the communication channel between server and client. [45] presents a mathematical model valid for channels with uniform channel losses. In the methodology applied in this chapter, a Markov model models the channel packet losses, in order to account for the characteristic burstiness of wireless communication channels. The next subsection (Section 4.3.1) describes the model for carousel retransmissions, where the expected number of cycles is derived from the expected number of packets received every cycle. Section 4.3.2 describes how the Markov model is applied to the calculation of the expected number of packets received per cycle. Finally, Section 4.3.3 presents an algorithm to calculate mathematically the download time.

4.3.1. Analysis of carousel retransmissions

The probability of receiving new packets is different depending on whether or not the AL-FEC is applied. When No-FEC is used, the probability of receiving x new packets at any given loop can be modeled by a hypergeometric distribution:

$$\mathrm{P}(x,m,n,l) = \frac{\binom{m}{x}\binom{n-m}{(n-l)-x}}{\binom{n}{n-l}}, \tag{4.1}$$

where *n* is the number of symbols or packets (as there is no FEC, *n* = *k*) of the file, *l* is the number of lost packets per loop and *m* represents the number of missing packets at the beginning of the loop. The denominator expresses the probability of receiving *n* - *l* packets in a carousel cycle. Similarly, the numerator expresses the probability of receiving exactly *x* new packets of the *m* missing packets out of the received *n* - *l* packets.

The range of all possible values for *x* is [0, *m*]. Thus, the latter probability yields the following expression for the expectation value of the number of packets correctly received at loop *i*:

$$x(i) = \sum_{\xi=0}^{m} \xi \cdot \mathrm{P}(\xi,m,n,l). \tag{4.2}$$

The number of cycles needed to download a file can be estimated using the following equation:

$$n_{cycles} = \min\left\{ h : \sum_{i=1}^{h} x(i) = n \right\}. \tag{4.3}$$

Similarly, if AL-FEC is used, the probability of receive *x* new packets at any given loop is defined by

$$\mathrm{P}(x,n,r,l) = \frac{\binom{n-r}{x}\binom{r}{(n-l)-x}}{\binom{n}{n-l}}, \tag{4.4}$$

where, in this case, *n* is the number of encoding symbols (source symbols plus parity symbols), *r* is the number of received symbols at the beginning of the loop and *l* is the number of lost packets per loop. It should be noted that this expression is equal to (4.1), making the substitution *m* = *n* - *r*. In this case, the expectation value is defined by

$$x(i) = \sum_{\xi=0}^{n-r} \xi \cdot \mathrm{P}(\xi,n,r,l). \tag{4.5}$$

Then, an estimation of the number of cycles is provided by the following expression:

$$n_{cycles} = \min\left\{ h : \sum_{i=1}^{h} x(i) \geq k * inef\_ratio \right\}. \tag{4.6}$$

It is important to remember that in order to rebuild a file successfully, clients have to receive a number of packets slightly higher than the number of packets that compose the file. This condition is reflected by considering the inefficiency ratio, as explained in Chapter 2.

4.3.2. Markov model channel

The Markov model [57], widely used in the literature, simulates well the burst losses, typical in wireless networks. Specifically, the two-state Markov model (also known as Gilbert model) establishes that the probability of losing a packet depends on whether the previous packet has been received or not, as Fig. 4.2 shows. Thus, in a bursty wireless channel, it is more likely to lose a packet if the previous packet is lost ($1 - q > p$).

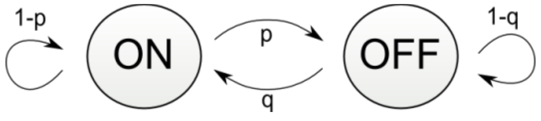


Fig. 4.2. State transition diagram for an example simplified Gilbert model.

The main parameters that characterize a lossy communication channel are the average loss probability (P_{loss}) and the average burst size (b). An appropriated configuration of the parameters p and q allows to model a channel with a given loss probability and burst size:

$$P_{loss} = \frac{p}{p+q}, \qquad b = \frac{1}{q}. \tag{4.7}$$

Fig. 4.3 depicts a state transition diagram defined by applying this model to the FLUTE transmission with carousels and considering that in each cycle of the carousel the server sends n packets. Each state in the diagram contains a pair (x, y) of numbers, where x is the number of packets received in the current loop, and y indicates if the last packet was received (0-ON) or not (1-OFF). Thus, there will be $2n + 1$ possible states in the transition diagram.

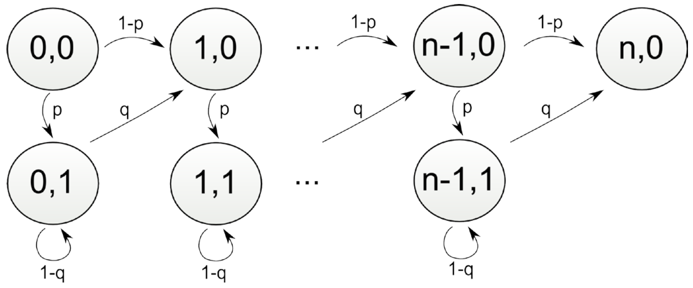


Fig. 4.3. State transition diagram for the Markov model in the transmission of *n* packets.

The transition matrix associated, with dimensions [(2*n* + 1) x (2*n* + 1)], is

$$T = \begin{pmatrix} 0 & p & 1-p & 0 & 0 & 0 & \dots & 0 \\ 0 & 1-q & q & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & p & 1-p & 0 & \dots & 0 \\ 0 & 0 & 0 & 1-q & q & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & p & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-q & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1-p \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & q \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}. \tag{4.8}$$

The probabilities vector is a [2*n* + 1] vector that indicates the probability of being in each state:

$$\Pi(i) = \left(P_{0,0}\ P_{0,1}\ P_{1,0}\ P_{1,1}\ P_{2,0} \cdots P_{n,0}\right). \tag{4.9}$$

Therefore, the probabilities vector in the iteration *i* (that is, after *i* packets have been sent) is

$$\Pi(i) = \Pi(0)T^i, \tag{4.10}$$

where *T* is the transition matrix and Π(0) is the initial probabilities vector. Considering that initially the system is in the state ON, since at the beginning no packets have been received, Π(0) is

$$\Pi(0) = (1\,0\,0\,0 \cdots 0). \tag{4.11}$$

In this way, the average number of packets received after *i* iterations is calculated adding up the number of packets of each state multiplied by the probability of being in each state:

$$\overline{N}(i) = \sum_{\eta=0}^{i<n} \eta \cdot \left[P_{\eta,0}(i) + P_{\eta,1}(i) \right] + n \cdot P_{n,0}(i). \tag{4.12}$$

The total number of packets received in a loop is provided by applying ($i = n$) in (4.12). Hence, the estimated number of lost packets per cycle is equal to

$$l = n - \overline{N}(n). \tag{4.13}$$

With this value, the number of cycles needed to download a file is calculated using formulas (4.1)-(4.6).

4.3.3. Analysis of adaptive LDPC

As explained above, when using adaptive LDPC, clients will receive the file with No-FEC parity until they are able to join their corresponding parity channel. Thus, in order to model adaptive LDPC there is a need to combine the two methods described above, corresponding to the cases where AL-FEC is used and where it is not. This section presents an algorithm that performs such a combination to calculate the average download time using adaptive LDPC codes.

Basically, the algorithm hereby proposed (Algorithm 4.1) calculates the number of file packets downloaded by applying the method presented above for No-FEC for every cycle up to feedback time. If the file download is not finished at feedback time, then the algorithm applies the method for AL-FEC.

The algorithm uses the formulas presented in the previous sections, using the following input parameters: the values p and q from the Markov loss model, the number of packets that make up a file without FEC (k), the transmission rate (b), the packet size (S) and the feedback time (t_fd).

Moreover, in order to implement adaptive LDPC, Algorithm 4.1 has also as input parameters the code rate used to do the coding process and the inefficiency ratio derived from this code rate. These are two key parameters in the performance of the algorithm. Optimum values of code rate provide the minimum values for the download time. In this sense, the value of the inefficiency ratio is strongly dependent on the code rate, as we analyzed in Chapter 3. As mentioned, the inefficiency ratio depends on the type of coding, and low values of inefficiency ratio reduce the download time.

It is worth mentioning that it is necessary to take into account that the download can finish during a cycle, that is, before all packets of a file have been sent. For this reason, dichotomy Algorithm 4.2 and Algorithm 4.3 adjust the download time, obtaining the percentage of the last cycle in which the download has finished. This adjustment can have a great impact on the download time if the cycle time is very high. Specifically, Algorithm 4.2 adjusts the download time when No-FEC is applied (used in part 1 of the main algorithm) and Algorithm 4.3 adjusts it when adaptive LDPC is applied (used in part 2 of the main algorithm). Algorithm 4.2 and Algorithm 4.3 have as input parameters those that appear in equations (4.1) and (4.4), respectively. Algorithm 4.2 has also as input parameter the value of *m* from the previous cycle to the cycle where the download has finished, whereas Algorithm 4.3 has the *r* of the previous cycle. Moreover, Algorithm 4.3 has the inefficiency ratio as input parameter.

Note that, when several blocks are used it is necessary to modify the proposed algorithm. In the modified algorithm, it is needed to take into account that the feedback message will arrive when a specific block is being received. Hence, some blocks can be received without FEC, other blocks can be received without FEC and then with optimum LDPC, and still other blocks can be received only with optimum LDPC. Therefore, the last block downloaded will determine the download time.

Algorithm 4.1: Adaptive LDPC

```
INPUT: p, q, k, S, b, t_fd, coderate, inef_ratio
OUTPUT: download_time
1:  Initialize (num_cycles1 = 0, num_cycles2 = 0)
2:  Calculate in which cycle the feedback message (c_fd) arrives
Part 1: No-FEC
3:  while (not all packets have been received and num_cycles1 + 1 < c_fd)
4:      Calculate number of losses per cycle according to Markov model (p,q) using
        (4.13)
5:      Calculate new packets received (P) in the current loop using (4.2) and update
        the total number of packets received
6:      num_cycles1 = num_cycles1 + 1
7:  end
8:  if (all packets have been received)
9:      Obtain the percentage of the last cycle using Algorithm 4.2
10:     download_time = (num_cycles1 – 1 + percentage1) * k * S/b
Part 2: Adaptive LDPC
11: else
12:     while (not all packets have been received)
13:         Calculate number of losses per cycle using (4.13)
14:         Calculate new packets received (P) in the current loop with (4.5) and
            update the total number of packets received
15:         num_cycles2 = num_cycles2 + 1
16:     end
17:     Adjust download time obtaining percentage2 using Algorithm 4.3
18:     download_time = (num_cycles2 – 1 + percentage2) * k * S/b/coderate + t_fd
19: end
```

Algorithm 4.2: Download time adjustment (No-FEC)

```
INPUT: k, m, l, last_m
OUTPUT: percentage1
1:  Initialize (bottom = 0, top = 1)
2:  while (true)
3:      percentage1 = (bottom + top)/2
4:      Calculate new packets received (P) with (4.2) with input parameters:
        percentage1 * (k, m, l) and update total packets not yet received
5:      if (last_m – P < 0)
6:          top = percentage1
7:      else if (last_m – P > 0)
8:          bottom = percentage1
9:      else
10:         BREAK
11:     end
12: end
```

Algorithm 4.3: Download time adjustment (AL-FEC)

```
INPUT: k, r, l, last_r, inef_ratio
OUTPUT: percentage2
1:  Initialize (bottom = 0, top = 1)
2:  while (true)
3:      percentage2 = (bottom + top)/2
4:      Calculate new packets received (P) with (4.5) with input parameters:
        percentage2 * (k, r, l) and update total packets received
5:      if (last_r + P < k * inef_ratio)
6:          bottom = percentage2
7:      else if (last_r + P > k * inef_ratio)
8:          top = percentage2
9:      else
10:         BREAK
11:     end
12: end
```

4.4. Evaluation methodology

This section describes the methodology used to evaluate the performance of the proposed adaptive AL-FEC codes. The goals of the evaluation are to validate the analytical model presented above and to compare the performance of adaptive LDPC codes for content download services with other proposals. The metric selected for the evaluation is the download time. Thus, it is necessary to identify the system parameters that could affect the download time and define values for them that are relevant for the case under study, as Section 4.4.1 describes. Furthermore, in order to compare analytical and experimental results, it is necessary to setup a valid scenario for conducting trials with the adaptive LDPC implementation, which is explained in Section 4.4.2.

Since optimum LDPC is an ideal implementation of adaptive LDPC, it is necessary to obtain the AL-FEC rate that minimizes the download time in the evaluation scenario for every packet loss rate. These values are later used to compare the developed implementation with the lower bounds established by optimum LDPC.

Once all environment conditions are set and optimum LDPC is modeled, the evaluation will consist of comparisons of the download time achieved under different configurations of the system parameters.

4.4.1. Evaluation parameters

As mentioned, the parameter used in the evaluation is the average download time. The study consists of measurements of this time obtained by applying different AL-FEC codes to FLUTE file delivery sessions: No-FEC, optimum LDPC (Staircase and Triangle), adaptive LDPC and rateless codes.

The comparison between these codes is done analyzing their behavior in different environments. Specifically, this chapter evaluates these codes for different file sizes, different number of blocks, transmission rates and feedback times.

In this sense, the measurements consider two different file sizes: 3000 and 6000 packets file size (i.e., over 4 and 8 MB, as each packet contains 1428 bytes). These are typical sizes of multimedia contents played in mobile devices, such as music files or short videos [74].

Moreover, two different number of blocks have been used: 1 block and 10 blocks. In efficiency terms, it is more efficient to send the files using one block. However, the block represents the decoding unit and hence clients require less memory when they work with small blocks. Therefore, using several encoding blocks can be highly recommended when clients have limited resources.

Furthermore, two different transmission rates have been used: 5 Mb/s and 10 Mb/s. In addition, feedback times of 1, 3 and 5 seconds have been used, since these are reasonable response values according to the transmission rates used.

The results of the study are presented in Section 4.5, which contains two types of results: analytical and experimental. In the first, the evaluation is done calculating the download time through the algorithm presented in the previous section. On the other hand, the next subsection explains the performance of the experimental results.

4.4.2. Experimental scenario

The experimental results have been carried out in order to validate the analytical ones. Thus, a more exhaustive analysis of the different parameters can be made using the analytical model, since its performance is much faster and easier.

The performance of these experimental results has been carried out using the implementation of a FLUTE server and client developed by the authors (presented in Chapter 2), which implements all the aforementioned AL-FEC codes.

It is worth noting that the implementation of adaptive AL-FEC codes is not specifically regarded in the FLUTE standards. In order to implement the adaptive LDPC, this chapter proposes that the FEC information (i.e., the FLUTE header extension EXT_FTI) is included in all the parity symbols, so that clients detect the code rate as soon as they join the parity channel. The FLUTE RFC [2] does not establish the frequency or the type of packets that carry the FEC information, as the only requirement is that there is one packet with the EXT_FTI extension per file in a cycle and/or the FEC Object Transmission Information is included in FDT Instances.

Furthermore, the insertion of a new parity channel does not affect ongoing downloads. When the server decides to include a new parity channel, it generates the encoding symbols for the specific file without interrupting the base channel. Therefore, parity channels are only available after the server is able to process the file to generate the parity packets. Clearly, the complexity of the AL-FEC algorithm and the size of the block lengthen the time needed for a server to include the parity data in the scenario.

On the other hand, when a client joins a given parity channel, they keep the source symbols successfully decoded from the encoding symbols received in the base channel. However, if there is a change in the AL-FEC code rate, the client needs to discard previously received parity symbols, as these are no longer valid for the new code rate.

It is assumed that there is a feedback between server and client that provides the server with an estimation of the losses experienced by every client. A possible implementation of this feedback is described in [75].

Rateless codes are simulated according to their definition as near ideal FEC codes, which establishes that it is only necessary to receive a small additional percentage of the packets that make up a file to rebuild it, regardless the erasure rate of the channel [76].

As concluded in Chapter 3, it is also worth noting that the transmission scheme of the packets that compose a block affects the performance of LDPC codes. For this reason, the measurements apply a random transmission scheme (source and parity symbols are sent in a random order), which provides better results than a sequential scheme in the presence of burst losses [77].

In both analytical and experimental results, the measurements collect as many iterations as needed to provide a 99% confidence intervals. The measurements have been made in a controlled environment, simulating the losses in the channel with the two state Markov model. In order to simulate a typical wireless channel, different channel losses between 0% and 30% (in steps of 5%) have been simulated. Also a 50% losses channel has been simulated to see the general tendency in the different studies. Fixing a percentage of losses and an average burst size, parameters p and q from Markov model are obtained using equation (4.7).

In the encoding process, values of code rates between 0.2 (very strong protection) and 0.9 (weak protection) have been used, with a precision of 0.1. Bear in mind that code rate represents the relation between the source symbols of a file and the total encoding symbols, that is, k/n. Therefore, the less code rate, the more protection.

4.5. Results and analysis

This section presents the results of the average download time against different parameters such as the channel packet loss rate, the transmission rate, the file size or the number of blocks used to send a file.

Two main studies have been developed: the first evaluates the optimum coding and code rates in channels with different loss rates. Once these values have been obtained, adaptive LDPC is analyzed and compared with rateless codes and optimum LDPC codes.

The optimum codes and code rates are measured experimentally, according to the methodology described in the previous section. On the other hand, the evaluation of adaptive LDPC is done through analytical and experimental measurements.

4.5.1. Optimum codes and code rates

This study analyzes what is the optimum coding and the optimum code rate depending on the losses of the channel and different transmission parameters: the transmission rate, the content size and the number of blocks.

Just to give an example of the results obtained, Fig. 4.4 shows the download time of a 3000-packet file size, using 1 encoding block with a transmission rate of 5 Mb/s, applying LDPC Staircase codes. The results show that for every channel packet loss rate there is an optimum AL-FEC code rate that minimizes the download time. For instance, in channels with 25% of losses, the optimum code rate for LDPC Staircase is 0.7. These results are compared with the ones obtained with LDPC Triangle and Compact No-Code (no AL-FEC used). The best codes (Compact No-Code, LDPC Staircase or Triangle) and the best code rate for each percentage of losses are chosen as optimum.

In this sense, Table 4.1 shows the optimum codes and code rates obtained for each scenario. In the table, the AL-FEC codes are identified according to the numeric identifier assigned by the IANA: 0) Compact No-Code, 3) LDPC Staircase and 4) LDPC Triangle. The parameters of the four scenarios are as follows:

- **Case 1**: 3000-packet file size, 1 block, 5 Mb/s;
- **Case 2**: 3000-packet file size, 1 block, 10 Mb/s;
- **Case 3**: 6000-packet file size, 1 block, 5 Mb/s;
- **Case 4**: 3000-packet file size, 10 blocks, 5 Mb/s;

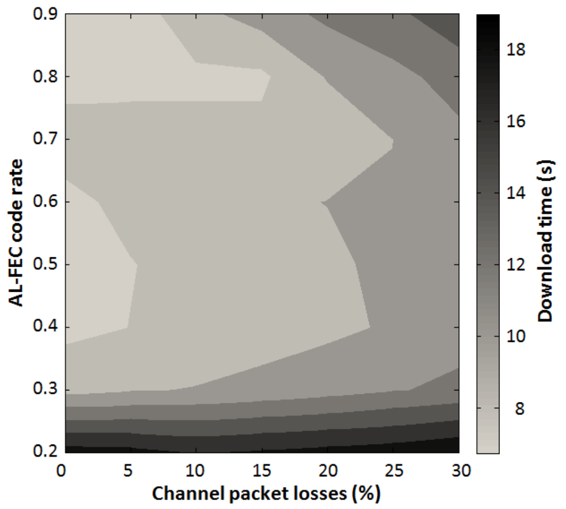


Fig. 4.4. Download time evaluation with LDPC Staircase codes with 3000-packet file size, 1 block and $b = 5$ Mb/s.

Table 4.1. Optimum coding parameters. IANA AL-FEC codes identifiers: (0) Compact No-Code, (3) LDPC Staircase, (4) LDPC Triangle.

Losses	**Case 1**	**Case 2**	**Case 3**	**Case 4**
0%	- (0)	- (0)	- (0)	- (0)
5%	0.9 (3)	0.8 (3)	0.9 (4)	0.8 (3)
10%	0.8 (3)	0.8 (3)	0.8 (3)	0.7 (3)
15%	0.8 (3)	0.7 (3)	0.8 (3)	0.6 (3)
20%	0.7 (3)	0.7 (3)	0.7 (3)	0.6 (3)
25%	0.7 (4)	0.6 (3)	0.7 (3)	0.5 (3)
30%	0.6 (3)	0.6 (3)	0.6 (3)	0.5 (3)
50%	0.4 (3)	0.4 (3)	0.5 (4)	0.3 (3)

Clearly, if there are no losses, the addition of AL-FEC parity penalizes the download time. For this reason, the best code for lossless channels is Compact No-Code in all scenarios. In the event of channel losses, LDPC Staircase generally provides better download times than LDPC Triangle. The optimum code rates range between 0.6 and 0.9 in most of the cases.

It should be noted that the optimum coding parameters are not only dependent on the channel losses, but also on other parameters like the file size, the number of blocks or the transmission rate. For instance, in the transmission of a 3000-packet file size, using 1 block, with a transmission rate of 5 Mb/s in a channel with 25% of losses, the optimum coding parameters are: LDPC Triangle with a code rate of 0.7. However, if the file is divided into 10 source blocks, the optimum coding parameters are LDPC Staircase with a code rate of 0.5.

Fig. 4.4 shows how, for every loss rate, the average download time increases as the code rate moves away from its optimal value. Nevertheless, moderate deviations of the actual channel packet loss rate from an estimated value may not increase the download time drastically. For instance, in the first scenario, using LDPC Staircase codes, the optimum code rate for a 15% packet loss is 0.8. The same code rate provides the best results for a 10% of packet loss and the results for 20% of losses are only slightly worse than for the optimum code rate of 0.7.

4.5.2. Evaluation of adaptive LDPC

In adaptive LDPC codes, the server changes the coding parameters (coding and code rate) upon reception of a message that informs about the losses of the channel after some feedback time. Once this message arrives, the server uses the results obtained in the previous study to choose the optimum coding parameters depending on the transmission parameters (losses, file size, transmission rate and number of blocks) and continues sending the file with the new parameters. In order to minimize the download time, the server continues sending from the last block that was being transmitted before the coding change occurred.

In the different studies it is assumed that, initially, the server sends the file using Compact No-Code (No-FEC) codes, so no protection is used.

As Fig. 4.5 shows, adaptive LDPC (A-LDPC) codes offer very good results compared to No-FEC. As the losses are higher, the need of using AL-FEC mechanisms is more obvious.

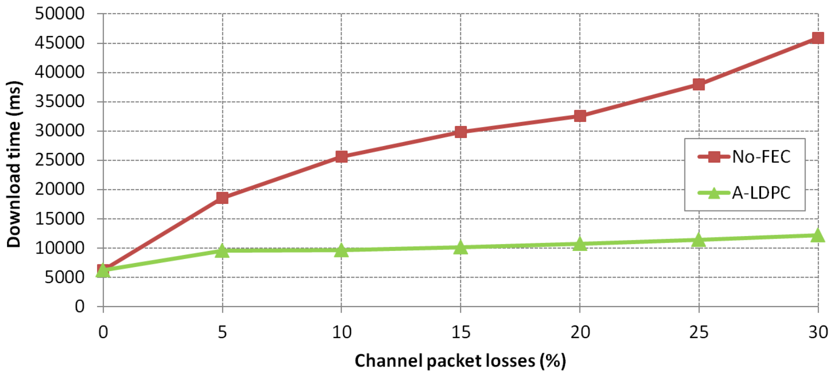


Fig. 4.5. Comparison between adaptive LDPC and Compact No-Code with 3000-packet file size, 1 block, $b = 5$ Mb/s and $t\_fd = 3$ s.

As mentioned, experimental results have been carried out to validate the analytical results. In order to see the differences between the analytical and the experimental results regarding adaptive LDPC codes, Fig. 4.6 shows a comparison between two files of different size: 3000 packets and 6000 packets.

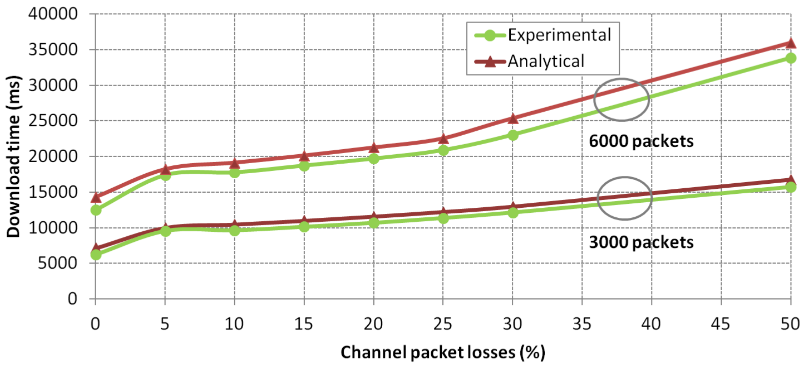


Fig. 4.6. Comparison between analytical and experimental results in a file size evaluation with 1 block, b = 5 Mb/s and t_fd = 3 s.

As Fig. 4.6 shows the values of analytical and experimental performance are very similar, although analytical results are slightly higher than experimental ones. This is due to the fact that analytical results use a fixed inefficiency ratio of 1.07 which it is not exactly the inefficiency ratio of each percentage of losses. As analyzed in Chapter 3, the inefficiency ratio depends on each coding and different transmissions of the same file can provide different values so, in some codes it is not possible to obtain the inefficiency ratio analytically. We have chosen the value of 1.07 according to the results obtained in the previous chapter and [51].

It is important to emphasize that all the studies hereby presented have been carried out analytically and experimentally. Since both models provide very similar download times, only the experimental results are shown. Nevertheless, in the different graphs, the experimental results include an upper error bar that represents the difference with respect to the download time obtained in the analytical results.

Returning to the file size analysis, Fig. 4.7 shows a comparison between the proposed adaptive LDPC (A-LDPC) codes, optimum LDPC (O-LDPC) and rateless codes. As expected, in all codes the download time of 6000-packet file size is approximately twice the download time of 3000-packet file size. For instance, the download time using adaptive LDPC codes with 20% of losses is 10 693 ms with 3000-packet file size and 19 710 ms with 6000-packet file size.

The difference between adaptive LDPC and optimum LDPC is lower as the file size is larger. On the contrary, the difference between optimum LDPC and rateless codes is higher. Nevertheless the download time ratio (the download time of optimum LDPC divided by the download time of rateless codes) gets better, so the larger the file size, the better the download time ratio.

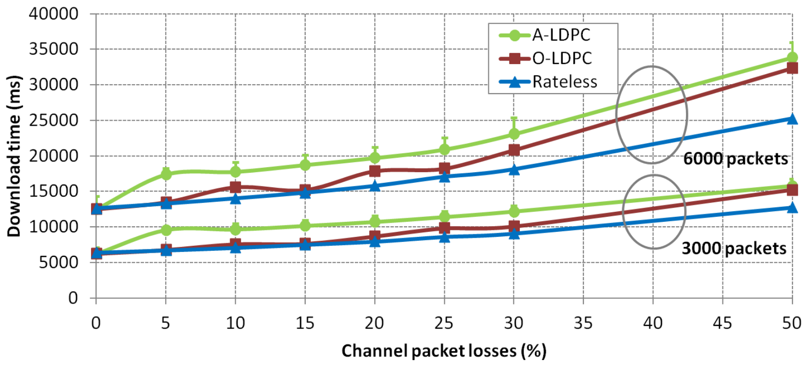


Fig. 4.7. File size evaluation with 1 block, b = 5 Mb/s and t_fd = 3 s.

On the other hand, Fig. 4.8 shows the behavior of adaptive LDPC compared with optimum LDPC and rateless codes for different feedback times. Regarding optimum LDPC codes, adaptive LDPC offers a good behavior, slightly worse than optimum LDPC if the feedback time is sufficiently short. As shown in the figure, the feedback time has a significant impact on the download time. In channel with high losses, the difference between the three feedback times (1, 3 and 5 s) decreases.

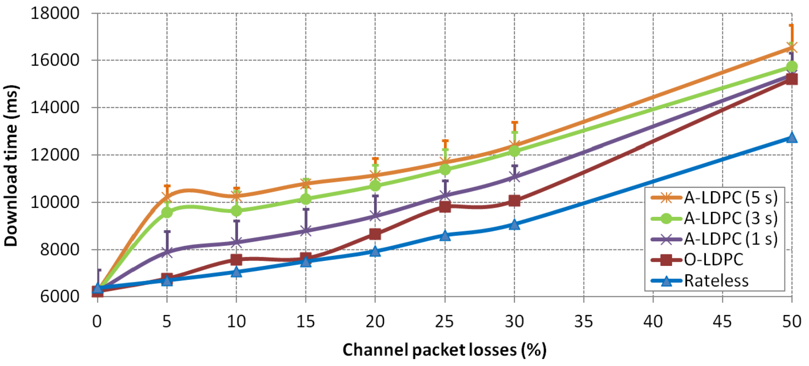


Fig. 4.8. Feedback time evaluation with 3000-packet file size, 1 block and b = 5 Mb/s.

The graph also shows that optimum LDPC codes perform very close to rateless codes (especially with moderate channel losses). In this sense, Fig. 4.9 shows the download time ratio with respect to rateless codes. The graph shows that the download time for optimum LDPC is only between 5% and 15% higher than rateless codes. This ratio is only slightly worse for adaptive LDPC, especially when the feedback time is short (around 20% for 1 s). When the losses are higher, the download time ratio is similar for the different feedback times and optimum LDPC.

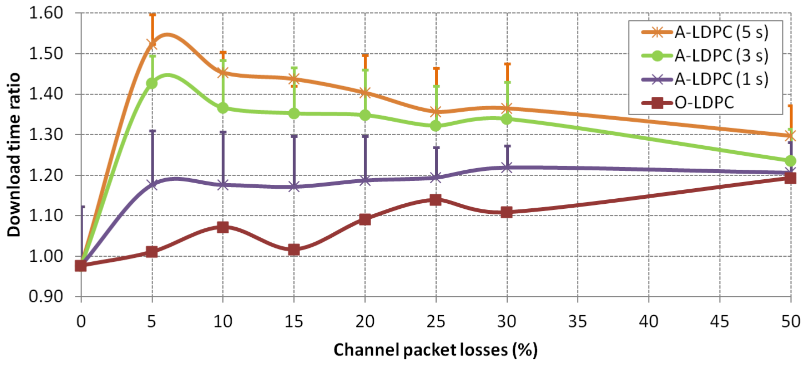


Fig. 4.9. Download time ratio with rateless codes with 3000-packet file size, 1 block and b = 5 Mb/s.

On the other hand, the transmission rate is, obviously, another parameter that affects the download time: the higher the transmission rate, the lower the download time. Fig. 4.10 shows that, when the transmission rate is doubled, the download time is approximately divided by two. For instance, using adaptive LDPC in channels with 30% of losses with a transmission rate of 5 Mb/s, the download time is equal to 12155 milliseconds, whereas the download time is 6815 milliseconds when the transmission rate is 10 Mb/s. With respect to adaptive LDPC, it is worth noting that for a fixed feedback time, the difference between adaptive LDPC and optimum LDPC codes is lower with low losses.

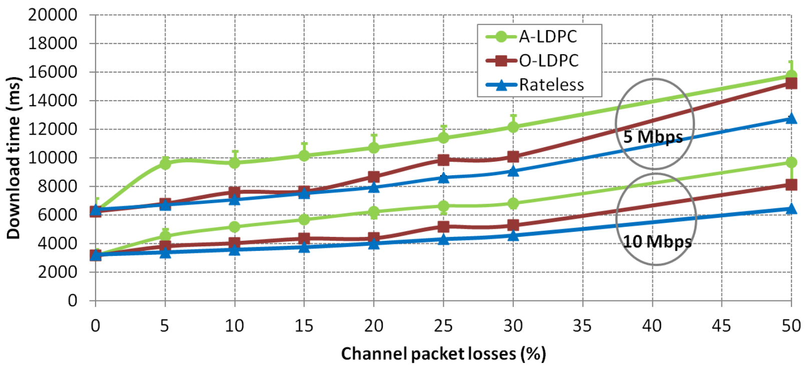


Fig. 4.10. Transmission rate evaluation with 3000-packet file size, 1 block and t_fd = 3 s.

Regarding the number of blocks, Fig. 4.11 shows how dividing a file into source blocks affects the download time. Both LDPC and rateless codes work more efficiently with large blocks. If more blocks are used, the download time gets worse for all AL-FEC codes.

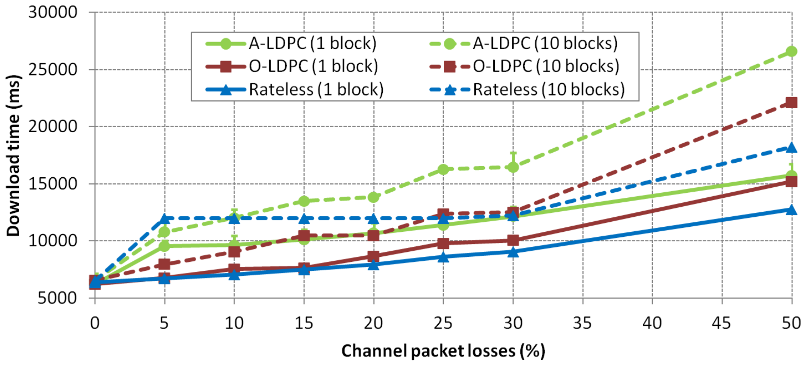


Fig. 4.11. Number of blocks evaluation with 3000-packet file size, b = 5 Mb/s and t_fd = 3 s.

It is important to note that optimum LDPC outperforms rateless codes when more than one block is used, in cases where the channel packet loss rate is relatively low (e.g. 5%). When several blocks are used, the download time is, in general, determined by the number of cycles needed to download the file. So similar percentages of losses involve similar download times if the number of cycles is equal. In this case, the last block that has not been downloaded determines the download time. So, if a certain block has not been decoded in the current cycle, it is necessary to wait one entire cycle to try to download the complete file.

Finally, emphasize that all the graphs hereby presented have shown similar results for the analytical and the experimental model. Hence, the analytical model proposed for adaptive LDPC AL-FEC codes is validated through experimental measurements.

4.6. Conclusions

This chapter has proposed the implementation of adaptive LDPC AL-FEC for multicast content distribution based on the FLUTE protocol. Adaptive AL-FEC codes represent a good alternative to improve the reliability of multicast connections over lossy channels, like wireless channels.

The different results show that it improves average download times to levels comparable to rateless codes, keeping the coding and decoding complexity of LDPC codes, but at the expense of an explicit feedback between clients and servers.

Despite that the ideal LDPC AL-FEC code rate depends on the amount of packets lost, a given AL-FEC code rate performs good in a wide interval of packet loss rates around the value for which it provides a minimum download time. In this way, it is expected that the performance of adaptive LDPC does not depend greatly on the accuracy of the channel packet loss estimation. However, results show that it depends considerably on the feedback time, defined as the time needed to provide clients with AL-FEC parity packets.

Although, in general, rateless codes offer better download times, different studies have shown that the ratios between rateless codes and optimum LDPC or adaptive LDPC codes are not very high. On the contrary, the decoding complexity is much lower in LDPC codes, which makes these codes highly recommended in receivers with limited resources, for instance, mobile devices. As mentioned, in these devices with limited resources it is recommended to send the data using several blocks, in order to reduce the decoding complexity. In that case the behavior of adaptive LDPC codes is very similar, even better for some percentage of losses, than rateless codes, as the results have shown.

On the other hand, in channels with limited bandwidth it is recommended to use few parity channels. So it is necessary to choose dynamically the optimum code rate depending on the different feedback messages received by all the clients, in order to satisfy the major part of them. The way to choose this best code rate for all users is explained in the following chapter.

The results of this chapter have been presented in an international conference [C.7] and in an international journal [J.6].

Chapter 5

Adaptive codes for limited bandwidth channels

This chapter presents an adaptive mechanism for improving the content download in wireless environments with limited bandwidth. Specifically, the system proposed reduces the average download time of clients within the coverage area, thus improving the Quality of Experience, without increasing the bandwidth. To that extent, clients send periodically feedback messages to the server reporting the losses they are experiencing. With this information, the server decides which is the optimum AL-FEC code rate that minimizes the average download time for all clients within the coverage area, taking into account the channel bandwidth, and starts sending data with that code rate. The system proposed is evaluated in various scenarios, considering different distributions of losses in the coverage area. Results show that the adaptive solution proposed is very suitable in wireless networks with limited bandwidth.

5.1. Introduction

As shown in the previous chapter, there is an optimum code rate that minimizes the download time of a certain file by a client. This download time depends, among other parameters, on the channel losses perceived by each client. In this way, it is possible to transmit a file at an optimum code rate for each client in order to minimize the download time. To that extent, clients must be able to inform the server about the losses they are experiencing.

Nevertheless, in environments with limited bandwidth it could not be possible to send a file with different code rates, since the bandwidth would increase considerably. Thus, it will be highly recommended to send data at an optimum code rate that benefits the major part of the users. Since the losses perceived by the users could change quickly, this code rate should be chosen dynamically.

In this sense, this chapter presents an adaptive system where the server sends data at an optimum code rate for each time interval, according to the losses detected by the clients within the coverage area.

Specifically, the objective of this chapter is to reduce the average download time of clients when downloading contents. It should be noted that, one of the main goals of any service is to provide a good Quality of Experience to the users. In streaming services a good QoE is provided when users receive the video without interruptions, with high quality, and with the minimum waiting time. With regards to file transmission to multiple receivers, users have a good QoE when they receive files correctly and the download time is minimal. In this sense, the download time is a well-known QoE metrics for evaluating file multicast download, for instance, in IP Datacast services.

The rest of the chapter is structured as follows. Next section provides an overview of the adaptive system proposed. Section 5.3 explains the evaluation methodology used to obtain the results presented in Section 5.4, where the adaptive system is analyzed and evaluated. Finally, the last section of the chapter includes some final conclusions.

5.2. System overview

This chapter proposes an alternative to the study presented in Chapter 4. Specifically, comparing to Chapter 4, this chapter presents a similar proposal which is more efficient in terms of bandwidth. It should be noted that in scenarios where the bandwidth is limited, it is necessary to find solutions that benefits all users. In the scenario proposed in this chapter, only one code rate is used in a given time, therefore only one transmission channel is needed. Thus, the file delivery session only contains one channel, in which both source and parity symbols are sent, which represents an easy solution for the server and the clients.

It is worth mentioning that, as in the previous studies, it is used a random transmission model, where source and parity packets are sent in a fully random order. Furthermore, this chapter considers that the losses perceived by the clients can change over time. We propose an adaptive mechanism to assign an optimum code rate similar to that used by RTP/RTCP (Real-time Transport Protocol / RTP Control Protocol) for dynamic adjustment of the bandwidth requirements of multimedia applications [78]. As in [78], the proposed algorithm increases, holds or decreases a certain parameter (in this case the code rate, instead of the bandwidth) according to the feedback received by the clients.

Fig. 5.1 shows an overview of the system proposed. There is a certain number of clients within the coverage area in a multicast wireless network. Clients perceive different losses depending on how far they are from the server. Also, clients are continuously moving so the channel losses they perceive are changing. Initially, the server sends data with a certain code rate. After a while clients start sending feedback messages informing the server about the losses they are perceiving. In this chapter we consider that the feedback messages always arrive to the server. The way clients send these losses reports is not analyzed in this chapter. Different mechanisms to provide this feedback are: [75] and [79], based on RTCP; and the reporting mechanisms used by DVB-H [9] and MBMS [33], which support FLUTE. Once the losses reports are received by the server, it decides which is the optimum code rate that minimizes the average download time of all clients. Then, the server starts sending data at this optimum code rate. This process is carried out periodically: the clients are repeatedly sending feedback messages and the server is analyzing them at a certain time intervals.

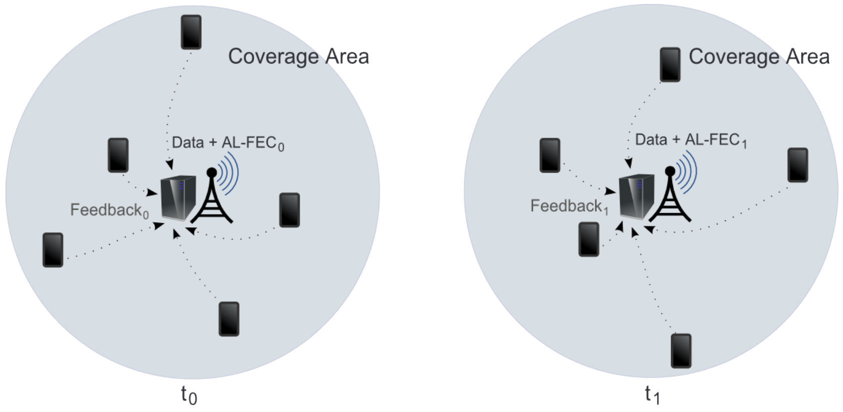


Fig. 5.1. System overview.

When choosing the code rate that best suits all clients it must be taken into account that using insufficient protection for clients with high losses has more impact on the download time than using an excessive protection for clients with low losses, as the results presented in Chapter 3 have shown. Therefore, feedback messages of clients with high losses will have more weight when the optimum code rate is chosen. To that extent, the server classifies the losses perceived by each user into three different regions: low losses, medium losses and high losses region. Hence, each one of the *n* clients is classified in a certain region according to their loss rate, as Fig. 5.2(a) depicts. The server calculates how many clients (n_L) are in the low losses region (clients who have less than λ_L losses), how many clients (n_M) are in the medium losses region (those who have between λ_L and λ_H losses) and how many clients (n_H) have high losses (those who have more than λ_H losses). Then, the code rate is chosen according to the percentage of clients in each region.

It is important to remember that, according to the conclusions obtained in Chapter 4 (presented in [80]), a given AL-FEC code rate performs well in a wide interval of packet loss rates around the value for which it provides a minimum download time. Based on this conclusion, a priori, it could be considered the use of only three different code rates (as many as losses regions). Nevertheless, at the time of choosing the code rates for each region, it must be taken into account the bandwidth increase associated to each code rate. Using a high protection (low code rates) increases considerably the bandwidth, therefore there is a trade-off between the bandwidth and the download time. According to the studies presented in Section 5.4.1, this chapter only considers two different code rates: one code rate for low losses and another one for medium-high losses. Using two code rates instead of three provides very good results regarding the download time, and improves considerably the channel bandwidth.

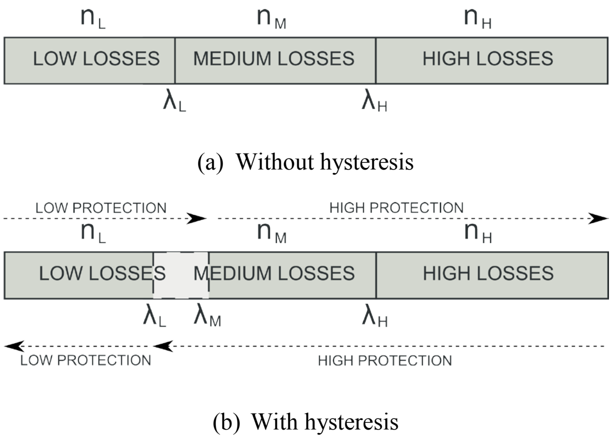


Fig. 5.2. Losses region classification.

In this sense, the use of a high number of losses regions would increase the complexity of the system and could make the system inefficient, since there would be a lot of changes of code rates and clients would have to create continuously the decoding parity matrix associated to that code rate and discard continuously parity packets previously received, as we will see in Section 5.4.4.

Hence, in our proposal there are two protection states: a state of low protection (which uses a high code rate) and a state of high protection (which uses a medium code rate). Furthermore, the encoding (and decoding) process is easier as less different code rates are used. The server calculates the optimum code rate in each instant of time according to the Algorithm 5.1, based on the one shown in [78] for bitrate:

Algorithm 5.1: Code rate calculation (2 states)

```
INPUT: n, $n_M$, $n_H$, $N_m$, $N_h$
OUTPUT: new_state
1:  if        $n_H/n \geq N_h$              then   new_state = HIGH_PROTECTION
2:  else if   $(n_M + n_H)/n \geq N_m$      then   new_state = HIGH_PROTECTION
3:  else                                    then   new_state = LOW_PROTECTION
```

N_h and N_m are the thresholds for clients with high and medium losses, respectively. The use of these thresholds allows to give more priority to clients with higher losses. The design of an efficient adaptive system depends greatly on the values of N_h and N_m. Other key parameters are λ_L and λ_H. In this sense, since it would not be very efficient to change the code rate too frequently when losses do not vary excessively, the system performance can be improved by using hysteresis [81]. Fig. 5.2(b) proposes the use of three thresholds: λ_L, λ_M, and λ_H. Therefore, when the server calculates the number of clients in each region, it takes into account the current protection state. Thus, if the server changes their state from the low protection state to the high protection state, a client in the low losses region will change to the medium losses region when their losses are higher than λ_M, whereas it will come back to the low losses region when their losses are lower than λ_L, as Fig. 5.2(b) depicts.

On the other hand, in order to avoid an erroneous estimation of the losses, the server can smooth, for each feedback message received, the instantaneous losses (L_{inst}) with the previous average losses (L_{avg}), using a low-pass filter, calculating the new loss rate (L) as: $L = (1 - \alpha) * L_{avg} + \alpha * L_{inst}$, where α is the influence factor of the new value, ranged between 0 and 1.

5.3. Evaluation methodology

5.3.1. Calculation of the download time

This section presents a methodology to calculate analytically the download time. As mentioned, the objective of this proposal is to reduce the average download time when clients download contents. The download time of a certain file *L* is defined as the time passed since the transmission starts until the file is completely downloaded. This occurs when clients have received enough packets to rebuild the file.

In this study we suppose that the server is sending files during a certain time, then it receives feedback messages from clients and then it continues sending files with the new code rate. For the sake of simplicity, in this theoretical study we consider that in each instant of time the server sends all files available in their repository. That is, this involves considering the use of file carousels to send content. In this way, we will consider that each cycle of the carousel corresponds to an instant of time.

Thus, as a first approximation, we are going to study the case where the server sends only one file. If a client is not able to download it during a certain instant of time, the client will need *T* instants of time to download the file. In each instant of time the server will send contents with a certain code rate, so the duration of each instant of time *i* ($t_C(i)$) could be different. Therefore the download time (t_D) is calculated as

$$t_D = t_S \cdot \beta + \sum_{i=1}^{T-1} t_C(i), \tag{5.1}$$

where t_S is the time needed to send all packets (source plus parity packets) that compose the file in the cycle that completes the download. Due to the use of AL-FEC encoding, clients can download a file before the last packets have been received so, actually clients could need a time lower than t_S to complete their downloads. To consider this, we define a factor $0<\beta\leq 1$. Remember that, in reception clients need to receive an amount of packets equal to the product of the number of source packets of the file to download by a factor called inefficiency ratio, which depends on the coding algorithm.

Developing expression (5.1):

$$t_D = \frac{ceil(\frac{S_L}{CR_T})}{b}\cdot\beta + \sum_{i=1}^{T-1}\frac{ceil(\frac{S_L}{CR_i})}{b}, \tag{5.2}$$

where S_L is the size of the file L to download, CR_i is the code rate of the instant of time i, and b is the transmission rate. Moreover, in order to calculate the number of packets that compose a file after decoding (and thus the transmission size) it is necessary to ceil the division between the number of packets that form a file by the code rate.

In the case of sending several files within the carousel, the calculation of the download time is slightly different. Fig. 5.3 shows an example of a transmission using file carousels. In the example we suppose that a client is connected to the channel at the start of a certain instant of time and that they want to download the file F3.

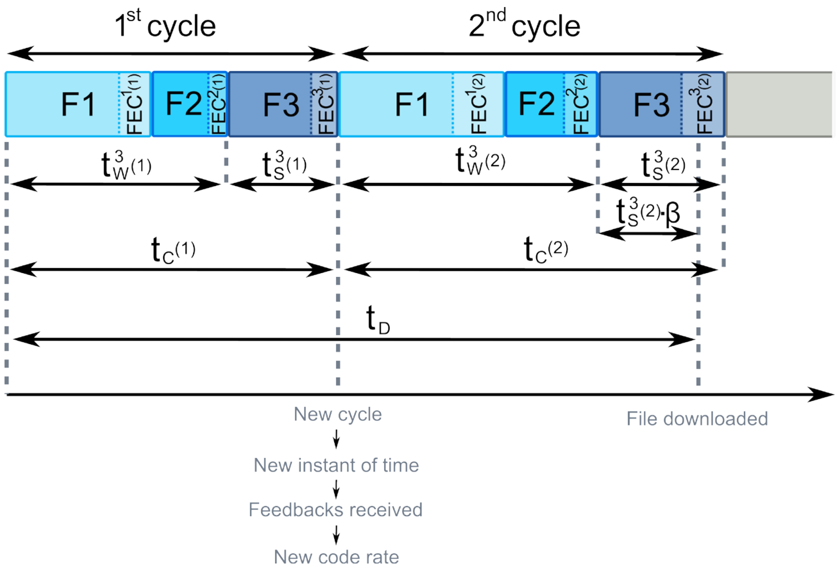


Fig. 5.3. Example of a carousel transmission.

First, if the client only needs one carousel cycle to download the file, the client will have to wait a time t_W to start receiving packets of the file L to download. After a time

$t_S \cdot \beta$, the client will have downloaded the file. If the client needs more than one cycle to complete the download, it will be necessary to consider the transmission time of the entire carousel (t_C).

In the example, each instant of time or cycle implies a different code rate so, as the protection in the second cycle is higher than in the first, the transmission size of every file in the second cycle is higher than in the first. Therefore, the transmission size of the entire carousel is bigger in the second cycle and thus the duration of that instant of time ($t_C\,(2) > t_C\,(1)$). The same applies to the value of t_W and t_S.

In this way, in the general case, clients will have to wait T - 1 entire cycles, plus a time t_W and t_S from the cycle that completes the download. So, the download time of a file L is calculated as

$$t_D^L = t_W^L(T) + t_S^L(T) \cdot \beta + \sum_{i=1}^{T-1} t_C(i). \tag{5.3}$$

Analyzing each term, the value of t_W depends on the carousel size and the transmission schedule. Considering that the probability of downloading a certain file is equal to the probability of downloading another file in the carousel, the waiting time can be calculated using the following approximation:

$$t_W^L(T) \approx \frac{t_C(T)}{2} - \frac{t_S^L(T)}{2}. \tag{5.4}$$

The calculation of t_S is similar to the equation (5.2). In order to simplify the final expression, we do not consider the effect of the ceiling round:

$$t_S^L(T) \cong \frac{S_L}{b \cdot CR_T}. \tag{5.5}$$

Regarding t_C, this term is calculated as

$$t_C(i) \cong \frac{\sum_{j=1}^{N} S_j}{b \cdot CR_i}, \tag{5.6}$$

where N is the number of files in the carousel and S_j is the size of the file j.

In this way, the download time is calculated as

$$t_D^L \approx \frac{\sum_{j=1}^{N} S_j}{2b \cdot CR_T} - \frac{S_L}{2b \cdot CR_T} + \frac{S_L}{b \cdot CR_T} \cdot \beta + \sum_{i=1}^{T-1} \frac{\sum_{j=1}^{N} S_j}{b \cdot CR_i}. \tag{5.7}$$

Simplifying that expression:

$$t_D^L \approx \frac{S_L}{b{\cdot}CR_T}\cdot\left[\beta-\frac{1}{2}\right]-\frac{\sum_{j=1}^{N}S_j}{2b{\cdot}CR_T}+\sum_{i=1}^{T}\frac{\sum_{j=1}^{N}S_j}{b{\cdot}CR_i}. \tag{5.8}$$

Analyzing the previous formula, values of S_j, N and b depend on each particular implementation. On the other hand, the value of the code rate for each instant of time (CR_i) is calculated using Algorithm 5.1, which will be analyzed in Section 5.4. The parameter β depends on the remaining packets needed to complete the download. This parameter is directly related to T, which is the number of instants of time (or number of cycles) that a user needs in order to download a certain file.

In order to obtain T, we use the methodology explained in Chapter 4, that is, calculating the number of new packets received by cycle and checking for each cycle if the client has received enough packets to rebuild the file. The calculation of this number of cycles is obtained using equations (4.4)-(4.6).

5.3.2. Evaluation parameters

As Section 5.2 has shown, there are several parameters to configure when evaluating the system proposed. Firstly, it will be analyzed the download time of a file in a channel with different losses using different code rates. In the studies hereby presented, in order to calculate which is the code rate that minimizes the download time for each percentage of losses two different methodologies will be used: first, the download time will be calculated using the analytical model explained in the previous subsection; second, we will calculate the download time by carrying out measurements in a real environment. To do this, it will be used again the file server and client based on FLUTE implemented in the context of this thesis. It should be highlighted that the major part of the results presented in this chapter are obtained through this implementation. Regarding the coding, in this chapter we use LDPC Staircase codes to evaluate the optimum code rate.

It is worth noting that both in the analytical and in the experimental results, several measurements are made. In the experimental ones, a server sends a file in a multicast channel and a client downloads it. Apart from getting the best code rate for each losses region, the previous study also will calculate the suitable values for λ_L, λ_M, and λ_H.

In the scenario proposed, there are many clients within the coverage area that are continuously moving (sometimes they are getting closer to the server and sometimes are moving away), so the losses they perceive are continuously changing. As mentioned, the losses they perceive are directly related to the distance to the server. In [82] it is shown the relation between the distance to the server and the PRR (Packet Reception Ratio) in a particular wireless network. Based on that study, Fig. 5.4 allows to match the distance from clients to the server with the percentage of losses that clients have.

These results have been carried out through simulations using ns-3, considering a transmission rate of 5.5 Mb/s and an output power of 17 dBm. This transmission rate is supported, among other standards, by 802.11b [83] (and subsequent versions), one of the Wi-Fi reference standards. Assuming that, in practice, the effective transmission rate is lower than the one specified in the standard (due to overheads or routers features), in the rest of studies an effective transmission rate of 5 Mb/s is used, the same considered in Chapter 4. All these studies only consider clients within the area lower than 110 meters, since distances higher than 110 meters provide unreasonable percentage of losses (higher than 60%), so it is considered that these clients are out of the coverage area.

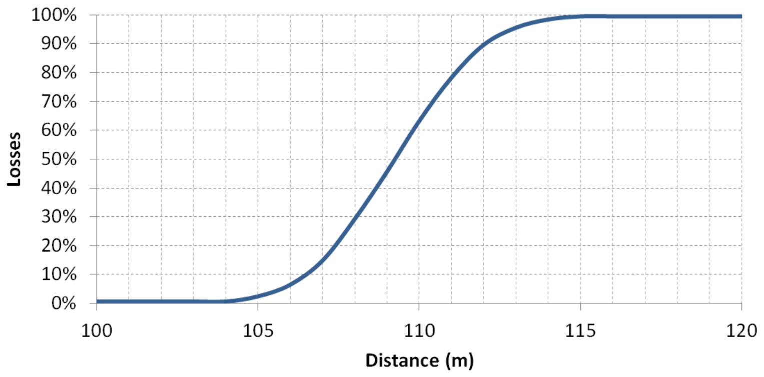


Fig. 5.4. Losses perceived depending on the distance to the server for a transmission rate of 5.5 Mb/s.

The percentage of losses according to the distance to the server depends on the transmission rate [84] as well as the transmission power. Thus, if the transmission rate increases (or the output power decreases), the coverage area decreases. For instance, with a transmission rate of 11 Mb/s, clients further away 20 meters from the server will not receive hardly any packet. Taking this into consideration, we will suppose that clients send their feedback messages using a lower transmission rate (for example, 1 Mb/s) with an appropriate output power through a channel that guarantees that the packets arrive to the server.

In order to analyze the behavior of the adaptive system proposed, the losses perceived by the clients change in every instant of time. Thus, there will be some moments when the average losses of all clients will increase whereas in other moments will decrease. The initial position of clients (and thus, losses) is generated randomly. Later positions of clients are calculated by using a simplified version of the Gauss-Markov mobility model [85]. This model, widely used in the literature, takes into account the previous position and speed of clients to estimate the future position of clients. In our case, the future position of clients is obtained by choosing a random value in a normal (Gaussian) distribution, which mean is the previous position.

Different losses scenarios are presented, as Section 5.4.2 explains. Those scenarios have a losses distribution completely different. Also, in all scenarios, clients are moving (and thus changing their losses). The system behavior is evaluated, in most cases, during ten time intervals.

Then, parameters N_h and N_m will be analyzed, as well as the smooth factor α and the hysteresis effect. Once all the configuration parameters are chosen, it is possible to calculate the download time for each client in each instant of time. This download time is calculated using the results obtained by means of the FLUTE server/client and by the analytical results. In order to do a more accurate measurement, the download time of a certain client in the instant i is calculated as the average of the download time between the instant i - 1 and the instant i, using the code rate obtained in the instant i – 1. Also, ten intermediate values in a time interval have been used to calculate the average download time. Also, for simplicity we have not considered co-channel interferences from other users.

5.4. Results and analysis

5.4.1. Analysis of the code rate

First, the optimum code rate for each percentage of losses is analyzed. To that extent, we calculate the download time of a file of approximately 4 MB (3000 FLUTE packets with length 1428 bytes), which is sent in a multicast channel using a transmission rate of 5 Mb/s. The download time is calculated by measuring the time passed since a client starts downloading a file until the file is completely downloaded. It should be noted that, apart from the losses, the download time depends greatly on the values of the file size and the transmission rate. However, as both parameters have a linear behavior regarding the download time [80], the conclusions arisen in this study remain valid independently of the value of the file size and the transmission rate.

Table 5.1 shows the download time (in milliseconds) for different code rates and percentage of losses for the experimental measurements. The table reflects that there is a code rate (highlighted in italics) that minimizes the download time depending on the losses. Based on the results of Table 5.1, Table 5.2 shows the average download time for the three losses regions: low, medium and high. We have defined the low losses region as the area where losses are between 0 and 10%; the medium losses region corresponds with the area where losses are between 10 and 30%; and the high losses region is the area where losses are higher than 30%. Table 5.2 also shows the bandwidth increase for each code rate. It is worth recalling that a low value of the code rate increases considerably the bandwidth.

Table 5.1. Download time (in milliseconds) for different code rates and percentage of losses.

CR/ Loss	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0%	8926	7826	7556	7946	8514	7557	6751	***6225***
5%	9388	8228	7927	8362	8723	7572	***6752***	18555
10%	9776	8704	8408	9349	8714	***7582***	9318	25622
15%	10600	9102	8923	9652	8595	***7625***	10847	29802
20%	11174	9596	9631	10034	***8739***	10129	12463	32598
25%	11396	10296	10467	***10045***	10196	11640	13399	37966
30%	12534	11033	11557	***11461***	11509	12818	15321	45880
40%	14560	13625	***12362***	13679	14463	15724	20252	60090
50%	16859	***15310***	15876	17502	18026	20656	25208	74105
60%	22182	***20225***	20880	22364	24403	27858	34538	101534

Table 5.2. Download time (in milliseconds) for different code rates and losses regions and bandwidth increase.

CR/ Loss	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Low	9363	8253	7964	8552	8650	***7570***	***7607***	16801
Medium	11096	9746	9797	10108	***9551***	9959	12270	34374
High	16534	***15048***	15169	16252	17100	19264	23830	70402
$\triangle$B	233%	150%	100%	67%	43%	25%	11%	0%

As expected, if no coding is used (that is, the code rate is equal to 1), the download time increases drastically when losses increase. When losses are low, code rates 0.8 and 0.9 provide the minimum download time. Calculating the average download time for low losses (Table 5.2), both 0.8 and 0.9 provide a similar value (only a difference of 0.5%). Therefore, as the bandwidth increase for code rate 0.9 is lower, this code rate will be chosen for the low protection state.

Regarding the medium losses region, the code rate with lowest average download time is 0.7. Finally, with very high losses, the code rate that minimizes the download time is 0.4.

Nevertheless, since in the scenario proposed in this chapter the bandwidth is a limited resource (and that is why only one transmission channel is used), it is not acceptable to use a code rate that increases excessively the bandwidth. In this sense, the code rate 0.4 provides the best results for high losses at the expense of increasing the channel bandwidth a 150%. Comparing the results obtained for high losses with code rates of 0.4 and 0.7, the download time with a code rate of 0.7 is only 15% higher than the one obtained with a code rate of 0.4 when losses are 50%, whereas when losses are 30%, the download time of 0.7 is barely 4% higher than the one obtained with a code rate of 0.4. The difference regarding the bandwidth increase is clear: 43% with a code rate of 0.7 against 150% with 0.4. Therefore, the code rate of 0.7 will be chosen when losses are both medium and high, as mentioned in Section 5.2.

Summarizing, in the low protection state, the server will send data using a code rate of 0.9, whereas in the high protection state, the server will use a code rate of 0.7.

In order to calculate the threshold values of the low and medium regions shown in Fig. 5.2(b), that is, λ_L and λ_M, it is needed to compare the behavior of the code rates (CR) 0.7 and 0.9. In this sense, Fig. 5.5 depicts the download time of both code rates for different percentage of losses.

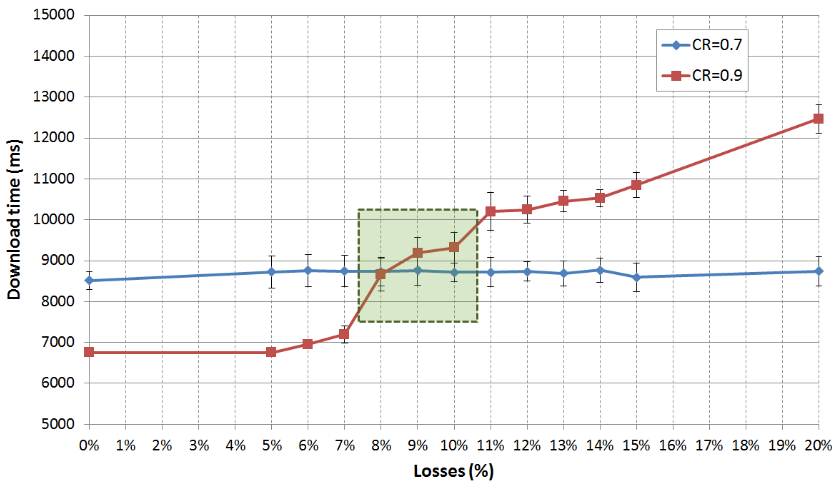


Fig. 5.5. Download time comparison between CR = 0.7 and CR = 0.9.

As Fig. 5.5 shows, when losses are equal or lower than 8%, the code rate 0.9 provides lower download times, whereas from 9% of losses the code rate of 0.7 is more suitable. Also, there is an area where the difference regarding the download time among the two code rates is very low, as Fig. 5.5 stresses. This area delimits the hysteresis zone. Initially, in order to provide a reasonable (not very tight) value of hysteresis, λ_L and λ_M will differ a 5%. Therefore, according to Fig. 5.5 and Table 5.1, the values of the thresholds will be: $\lambda_L = 7\%$, $\lambda_M = 12\%$ and $\lambda_H = 30\%$. These values will be analyzed in Section 5.4.7.

5.4.2. Losses model

The studies hereby presented consider that there are n = 100 clients in the coverage area. In order to analyze the behavior of the system proposed, five different scenarios with different distributions of losses are defined. In all scenarios, clients are continuously moving within the coverage area, with the aim of analyzing how the system works when losses change.

The distribution of the instantaneous losses of all clients for each instant of time of the first scenario is represented in Fig. 5.6. This scenario considers 10 instants of time. The figure shows the percentage of clients per losses region as well as the hysteresis region. Clients in the hysteresis zone will be in the low or in the medium losses region depending on the state protection. Moreover, the figure also shows the average losses perceived by all clients for each instant of time. On the other hand, Fig. 5.7 depicts, for the same losses distribution, the distance from clients to the server for all instants of time, and it can be clearly seen how clients are distributed along time. That figure also shows three circles that represent the losses thresholds λ_L, λ_M and λ_H. As mentioned, the amount of clients in each one of these circles will determine the state of the system and therefore the code rate.

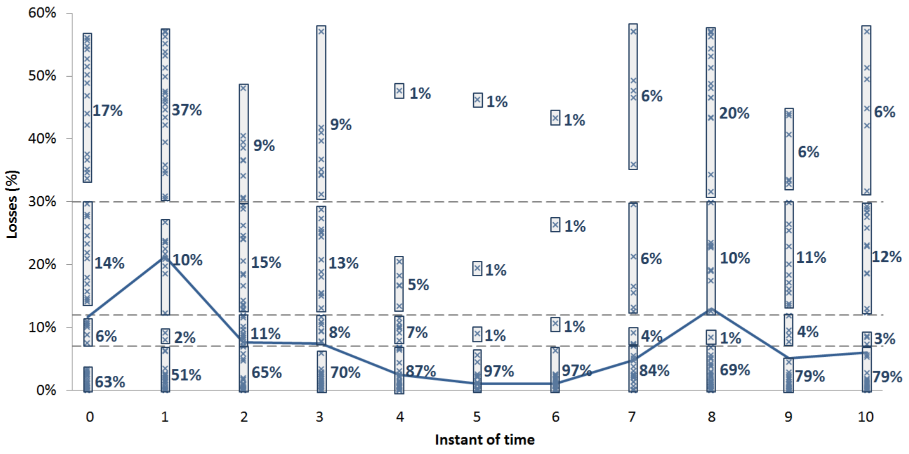


Fig. 5.6. Losses distribution for scenario 1.

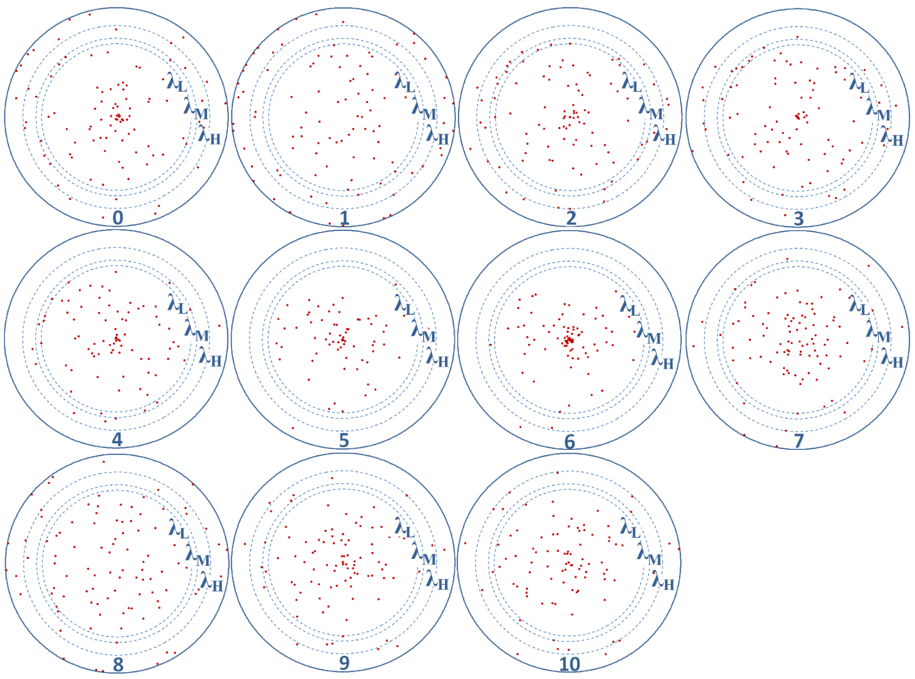


Fig. 5.7. Distance to the server for scenario 1.

In the second scenario, the distribution of losses is different, existing time intervals when clients get closer to the server and others when clients move further away from it, as Fig. 5.8 shows. In addition, in the third scenario (Fig. 5.9) the losses are, in general, rather higher.

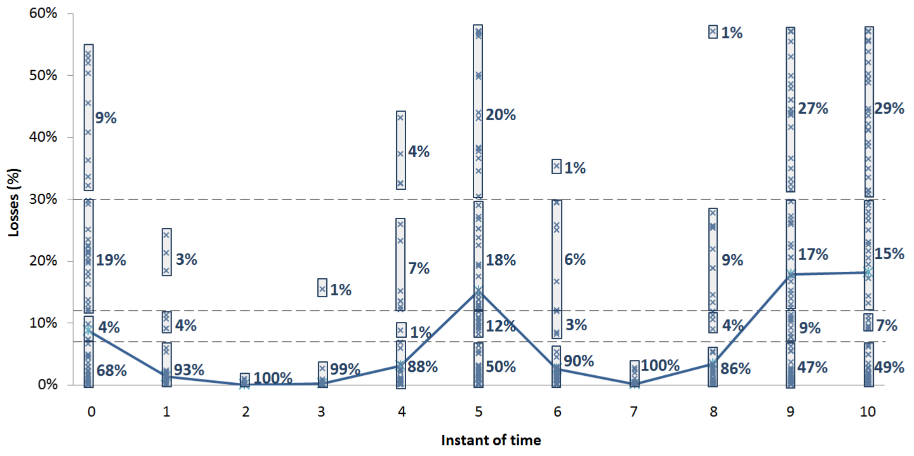


Fig. 5.8. Losses distribution for scenario 2.

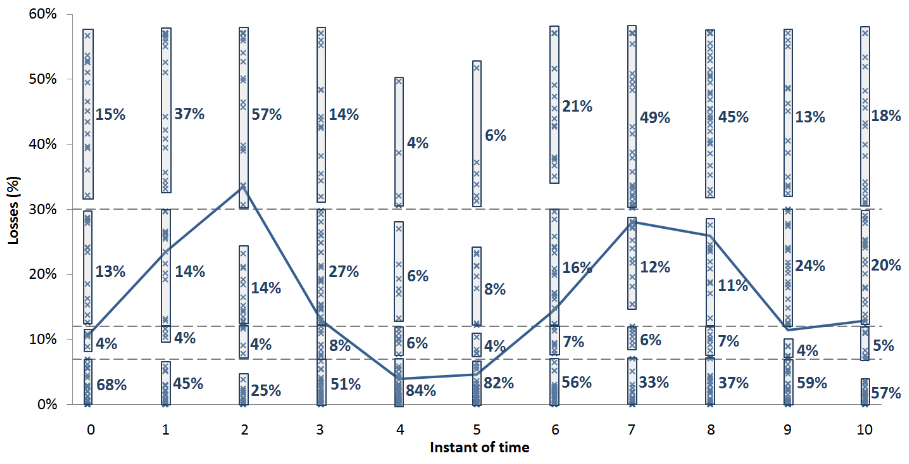


Fig. 5.9. Losses distribution for scenario 3.

On the other hand, the fourth scenario is rather different from the previous ones, since losses change more abruptly. Fig. 5.10 shows the distribution of losses for each instant of time as well as the average losses along the time.

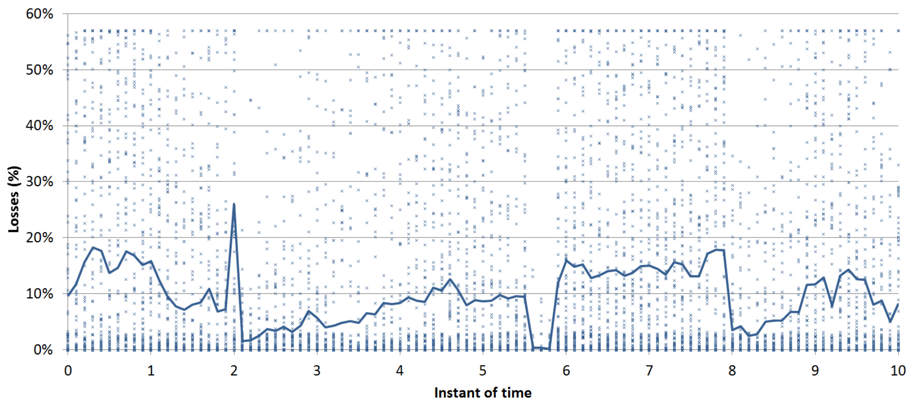


Fig. 5.10. Losses distribution for scenario 4.

Finally, the fifth scenario, shown in Fig. 5.11, will be used to evaluate the effect of the hysteresis. To that extent, the scenario considers many time intervals, specifically 100, instead of 10 used in the previous scenarios. Moreover, the average losses will be all around the hysteresis zone.

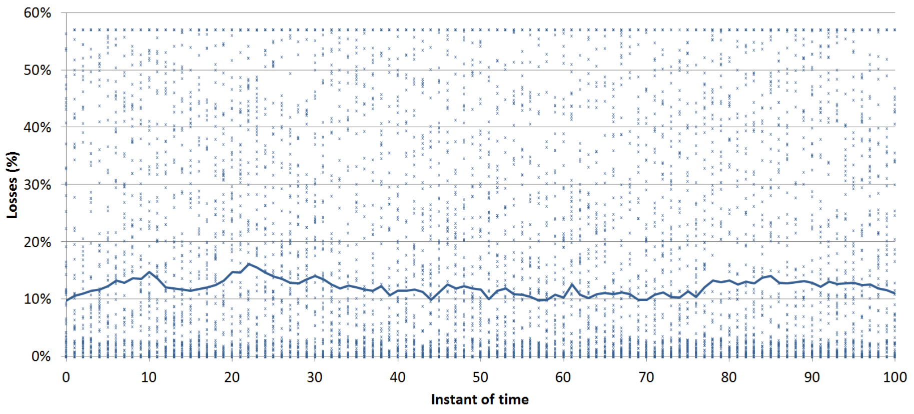


Fig. 5.11. Losses distribution for scenario 5.

5.4.3. Adaptive code rate

The value of parameters N_h and N_m has a great influence on the performance of the system proposed. As mentioned, it is necessary to give more priority to those clients who perceive high losses, therefore if a low percentage of clients has high losses the protection must be increased. In the following study, we consider N_h = 17% (1/6) and N_m = 33% (1/3), and later we will analyze other values. Therefore, if more than a sixth of clients have high losses or more than a third of clients have medium or high losses, the server will be in the high protection state. Also, initially we consider that $\alpha = 1$, so the server does not smooth the losses perceived by the clients. In the three scenarios used in this subsection this value seems appropriate since there are not excessively abrupt changes in losses.

As mentioned previously, this chapter presents both analytical and experimental measurements. In the figures shown, the analytical results are presented as an upper error bar that represents the difference regarding the download time obtained in the experimental results.

The performance of the adaptive system in scenario 1 is shown in Fig. 5.12. The figure shows the average download time per instant of time for different code rates: a fixed code rate of 0.7, a fixed code rate of 0.9, the adaptive code rate and the ideal case. This ideal case considers that each client connects to a channel which transmits with the optimum code rate according to their losses. That is, this case corresponds with the proposal presented in Chapter 4 after all parity channels (with code rates 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9) plus the base channel have been created. In this way, the ideal case provides the minimum average download time possible.

Initially (in the instant of time 0) the adaptive system is in the low protection state (CR = 0.9), and the adaptation begins in the next instant of time. Fig. 5.12 reflects how the adaptive system changes its protection state according to the client losses in each instant of time. As figure shows, the behavior of the adaptive system is rather good, since the server is sending data, in most cases, with the code rate that minimizes the download time. Apart from the initial instant of time, in the instant of time 8 the adaptive system is not transmitting with the optimum code rate. Nevertheless, in this case, the difference regarding the download time between code rates 0.7 and 0.9 is minimal, therefore in this scenario the adaptive system works almost perfect. Comparing with the ideal case, the adaptive system provides, on average, download times over 20% higher, which is a rather good result.

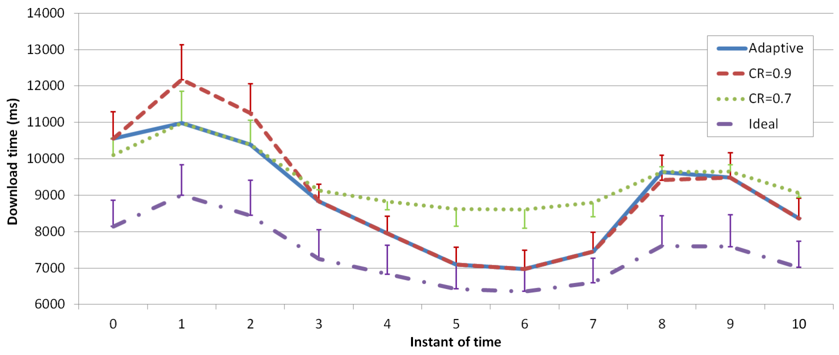


Fig. 5.12. Download time evaluation in scenario 1 for $N_h = 1/6$, $N_m = 1/3$.

The good behavior of the adaptive system is proven in scenarios 2 and 3, as Fig. 5.13 and Fig. 5.14 reflect.

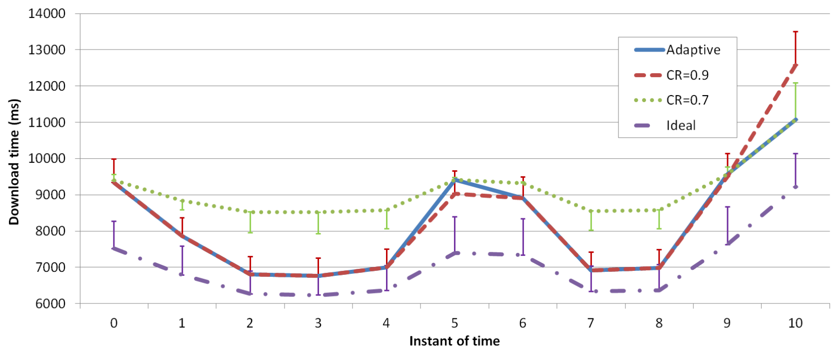


Fig. 5.13. Download time evaluation in scenario 2 for $N_h = 1/6$, $N_m = 1/3$.

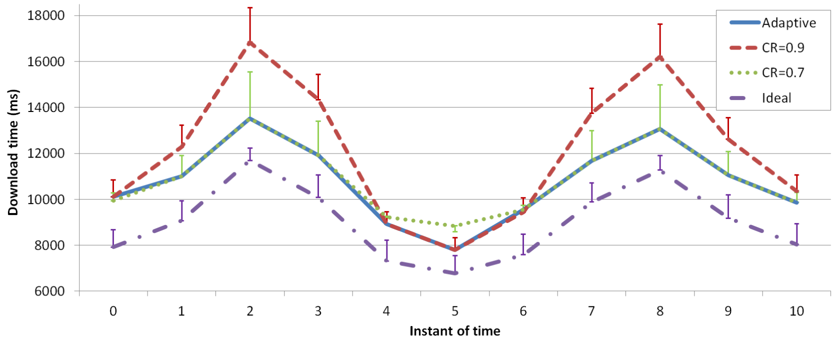
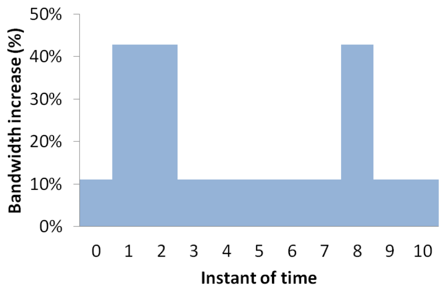


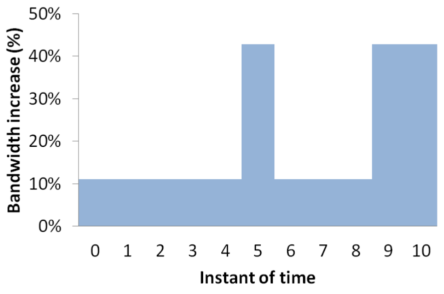
Fig. 5.14. Download time evaluation in scenario 3 for $N_h = 1/6$, $N_m = 1/3$.

Comparing the experimental and the analytical results in these three scenarios, we see that, in general, analytical results provide higher download times, but the difference is not very meaningful (it is not higher than a 10% in all cases). In fact, both experimental and analytical results provide the same adaptive protection state in all instants of times, so the analytical model works rather well. The differences among two models are due to the precision of the transmission rate module of the implemented file server when calculating the experimental results and due to the value of the inefficiency ratio used in the analytical results. It should be noted that the value of the inefficiency ratio cannot be calculated analytically in some codes (those which do not belong to the MDS category), since the inefficiency ratio depends on the order of the packets upon arrival. In this study we have used a specific value of the inefficiency ratio for each code rate, according to the results obtained in Chapter 3.

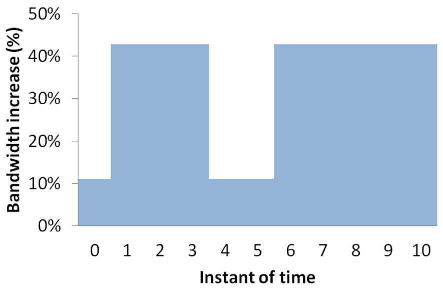
As mentioned, one of the most important parameters to take into account is the bandwidth increase due to the use of AL-FEC. As Table 5.2 has shown, the bandwidth increases over 11% with a code rate equal to 0.9 (in low protection state), whereas the increase of bandwidth for 0.7 (in high protection state) is over 43%. Thus, the average bandwidth increase in the adaptive system will depend on the protection state. In scenario 1 the average bandwidth increase is 19.8%, in scenario 2 is 19.8% too, and in scenario 3 is 34.2%. The bandwidth distribution per time intervals in the different scenarios is shown in Fig. 5.15(a-c). At this point, it is worth comparing the results obtained with the adaptive case regarding the ideal case. As mentioned, the ideal case provides, on average, a download time over a 20% lower than the adaptive case. Nevertheless, the ideal case provides an overhead of 396% (considering six parity channels plus the base channel).



(a) Scenario 1



(b) Scenario 2



(c) Scenario 3

Fig. 5.15. Bandwidth increase in different scenarios.

So far, we have considered that clients with high and medium losses have more weight than those with low losses. In order to analyze the effect of this condition, the next study gives the same priority to all losses areas, therefore the values of N_h and N_m change: N_h = 33% (1/3) and N_m = 67% (2/3). As a result, the server tends to be more frequently in the low protection state, as Fig. 5.16 (scenario 1) and Fig. 5.17 (scenario 2) show. Comparing with the ideal case, the difference regarding the download time among adaptive and ideal is approximately the same that in the previous studies, that is, over 20%.

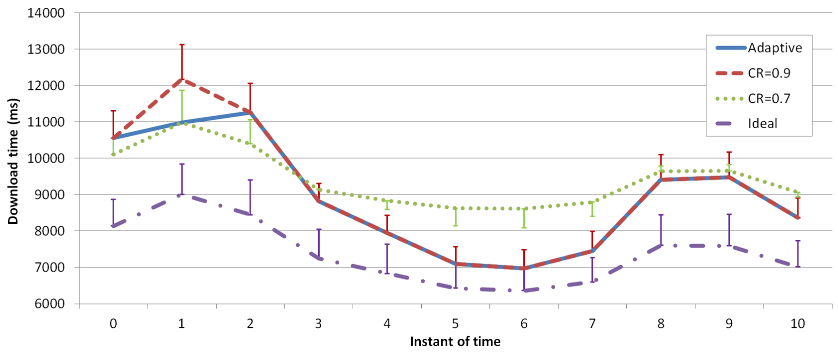


Fig. 5.16. Download time evaluation in scenario 1 for $N_h = 1/3$, $N_m = 2/3$.

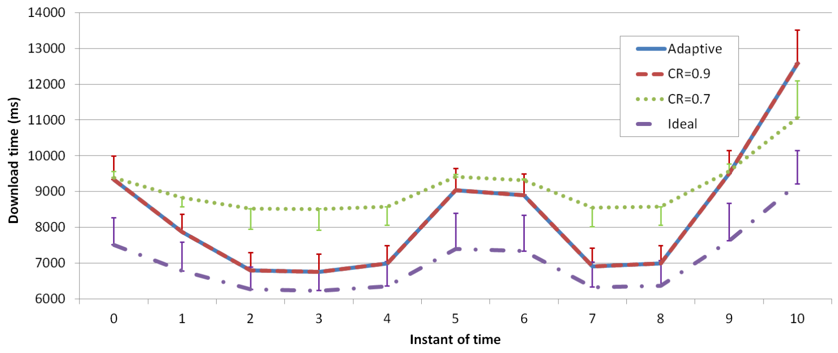


Fig. 5.17. Download time evaluation in scenario 2 for $N_h = 1/3$, $N_m = 2/3$.

Nevertheless, although in previous scenarios the adaptive system works rather well, in environments with a huge number of clients with high losses, previous values of N_h and N_m are not appropriate, such as in scenario 3, shown in Fig. 5.18.

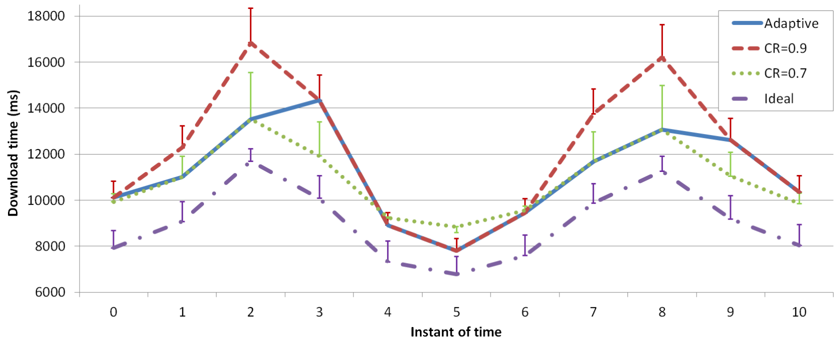


Fig. 5.18. Download time evaluation in scenario 3 for $N_h = 1/3$, $N_m = 2/3$.

Once again, both analytical and experimental results for the previous three study cases provide similar values, with a difference regarding the download time lower than a 10%.

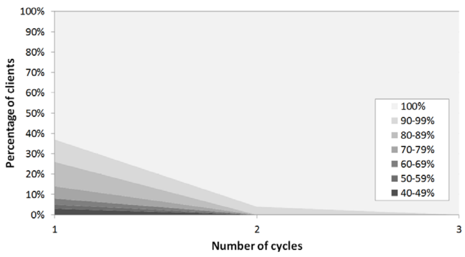
Finally, we study a particular case. We suppose that the server is sending the same file in a carousel, and each instant of time represents the moment when the server has sent the last packet of the file (that is, the end of the carousel). Thus, the time between two consecutive instants of time is the carousel period.

Initially all clients within the coverage area are interested in this file, so they start downloading it when the server begins to send data. Depending on the losses perceived by each client and on the code rate used to send the file, clients could need more than one transmission of the file (that is, various carousel cycles) to download it. After completing the download, clients leave the channel, so do not send more reports to the server. Therefore, the server only will consider those reports received by the clients within the coverage area for each instant of time. Next study analyzes the number of clients that have completed their downloads in each carousel cycle. In this case, scenario 2 has been used.

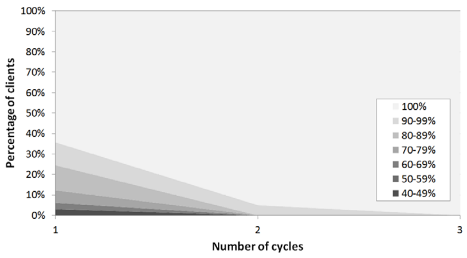
Fig. 5.19 depicts, for each instant of time, the percentage of clients that have download a certain percentage of the file, analyzing the cases of adaptive code rate –Fig. 5.19(a)– and code rates 0.9 –Fig. 5.19(b) – and 0.7 –Fig. 5.19(c). Obviously, as the protection increases, the number of cycles decreases. It is good to recall that, although using a high protection (low code rate) always decreases the number of cycles needed to download a file, this does not entail that the download time decreases, since the carousel size and the carousel period increases, as we have seen previously. Focusing on the adaptive graph of Fig. 5.19(a), we can see that after the first cycle period (instant of time 1), 63% of clients have downloaded completely the file, whereas a 3% have not downloaded even the half of the file (40-49%). One carousel cycle later, 96% of clients have completed their downloads.

On the other hand, each time the server changes the code rate, clients have to discard the parity packets previously received. Even so, as Fig. 5.19(a) shows, the adaptive code rate provides (slightly) better results than the code rate 0.9 –Fig. 5.19(b). The performance can be improved if clients, instead of discarding parity symbols previously received when the code rate changes, save them in case the server sends data with the previous code rate in future carousel cycles.

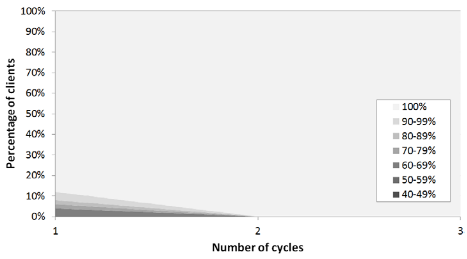
It is to be noted that considering that clients leave the channel once they have completed the download causes that the protection state changes. Thus, after the first carousel cycle the code rate changes from 0.9 to 0.7, whereas if consider that all clients remain within the coverage area, the code rate does not change during the first carousel cycles, as Fig. 5.13 showed.



(a) Adaptive



(b) CR = 0.9



(c) CR = 0.7

Fig. 5.19. Evaluation of the number of cycles to complete the download in scenario 2.

5.4.4. Multiple losses regions

In this section we analyze how the use of different losses regions would affect the results presented. For this study we define the double of losses regions, and one losses region for each code rate. The losses regions are defined according to the results presented in Table 5.1.

- **Losses region 1:** between 0 and 7% losses, using a CR = 0.9;
- **Losses region 2:** between 7 and 17% losses, using a CR = 0.8;
- **Losses region 3:** between 17 and 25% losses, using a CR = 0.7;
- **Losses region 4:** between 25 and 35% losses, using a CR = 0.6;
- **Losses region 5:** between 35 and 50% losses, using a CR = 0.5;
- **Losses region 6:** more than 50% losses, using a CR = 0.4;

The definition of these losses regions provides following thresholds (for simplicity hysteresis is not used): λ_1 = 7%, λ_2 = 17%, λ_3 = 25%, λ_4 = 35% and λ_5 = 50%. Furthermore, making an approximation of values of N_h and N_m used in previous studies, we fix following values: N_1 = 40%, N_2 = 25%, N_3 = 20%, N_4 = 15%, N_5 = 8%. Therefore, the algorithm is:

Algorithm 5.2: Code rate calculation (6 states)

```
INPUT: $n, n_2, n_3, n_4, n_5, n_6, N_1, N_2, N_3, N_4, N_5$
OUTPUT: new code rate (CR)
1:  if       $n_6/n \geq N_5$                              then   $CR = 0.4$
2:  else if  $(n_5 + n_6)/n \geq N_4$                      then   $CR = 0.5$
3:  else if  $(n_4 + n_5 + n_6)/n \geq N_3$                then   $CR = 0.6$
4:  else if  $(n_3 + n_4 + n_5 + n_6)/n \geq N_2$          then   $CR = 0.7$
5:  else if  $(n_2 + n_3 + n_4 + n_5 + n_6)/n \geq N_1$    then   $CR = 0.8$
6:  else                                                   then   $CR = 0.9$
```

where n_i represents the number of clients in the losses region i and n is the total number of clients within the coverage area.

Following figures show the code rate used (Fig. 5.20) and the average download time (Fig. 5.21) for each instant of time for the two cases to analyze: the original configuration shown in the previous section (that is, two different code rates– 2CR) and an adaptive system that uses six different code rates (6CR). The first study is carried out using 100 clients in the first losses scenario (Fig. 5.6). As results show, the use of several losses regions makes that the code rate changes continuously. Thus, using only 2 code rates there are 4 changes of code rate, whereas for 6 different code rates, for the time interval shown in Fig. 5.20, there are 8 changes of losses regions. However, the download time obtained for both configurations is very similar (Fig. 5.21), with an average download time reduction of a 2% using 6 different code rates. It should be noted that it could be possible to obtain better results regarding the download time if the thresholds (λ_i and N_i) used for each losses region are optimal, but it is rather difficult to adjust these thresholds when there are many losses regions. Even so, the results would not improve excessively. Finally, regarding the bandwidth, the use of two code rates produces an average overhead increase of 20%, whereas using six different code rates produces an overhead of 93%.

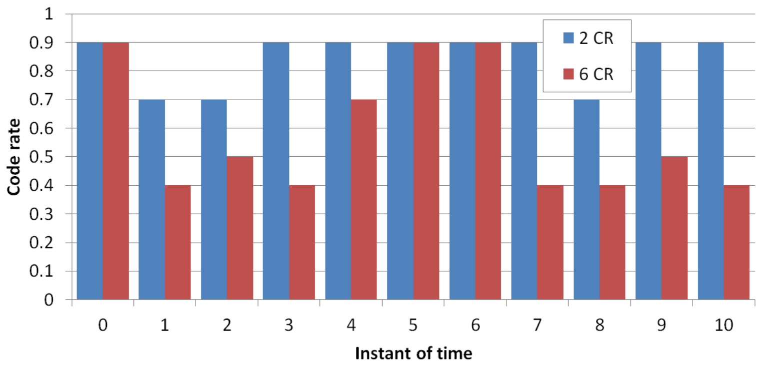


Fig. 5.20. Number of changes of losses regions in scenario 1.

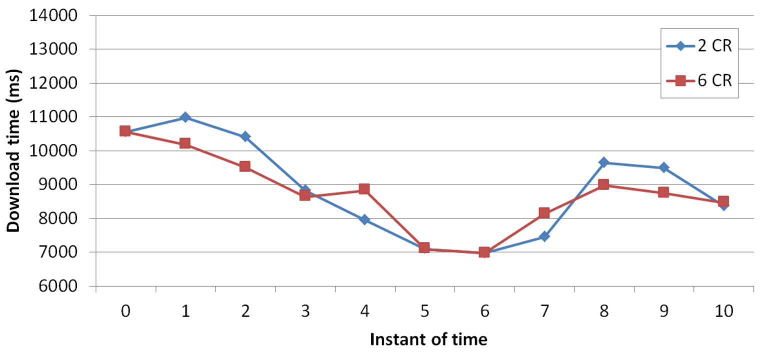


Fig. 5.21. Download time in scenario 1.

Results and conclusions obtained in scenario 2 are very similar. On the other hand, scenario 3 (shown in Fig. 5.9), where the average losses perceived by the clients are higher, provides different results. As Fig. 5.22 shows, both configurations provide the same number of changes of code rates. This is due to the fact that there is a high number of clients that perceive high losses, so both configurations provide the maximum protection possible (in one case 0.7 and in the other 0.4). Regarding the download time, the configuration with 6 losses regions provides better results for almost all instants of time (Fig. 5.23). However this configuration only reduces the average download time a 5%, at the expense of an average bandwidth increase of a 133% (whereas the configuration of only two code rates produces an overhead of a 34%). As mentioned previously, it is not worth using very much protection (for instance CR = 0.4) since this increases considerably the bandwidth, and there are other code rates (such as CR = 0.7) that provide a good trade-off between the download time and the overhead.

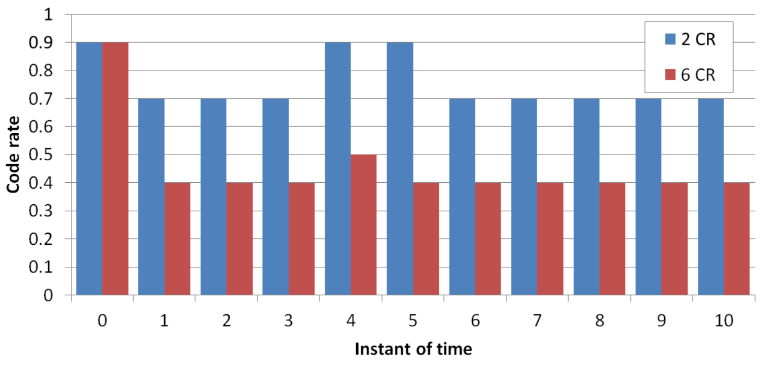


Fig. 5.22. Number of changes of losses regions in scenario 3.

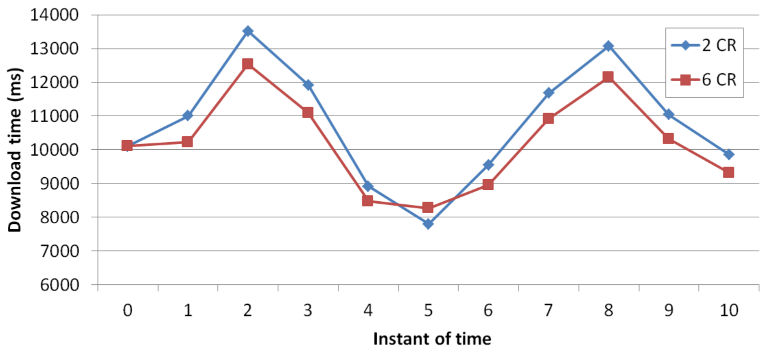


Fig. 5.23. Download time in scenario 3.

Therefore, results show that it is enough to consider three losses regions (and two code rates) to obtain a good performance of the system proposed.

5.4.5. Evaluation of the smoothing

Coming back to the analysis considering three losses regions, in networks where clients have high mobility, losses are continuously changing. Sometimes, when calculating the losses for a certain instant of time it can occur that the estimation is not correct, since there are peak errors that distort the average losses. In those cases, it is very common the use of a smooth factor, through which the server considers both the current and the previous losses in order to estimate the system losses. In the three scenarios previously shown, as there were not abrupt changes regarding the losses, no smooth process was used (and thus α was equal to 1). In order to evaluate how this smooth parameter affects the system proposed, next study considers scenario 4 (shown in Fig. 5.10). Fig. 5.24 depicts the distribution of the average losses of n = 100 clients along time as well as the smooth effect for different values of α. In this scenario some bursts appear, where losses change rapidly.

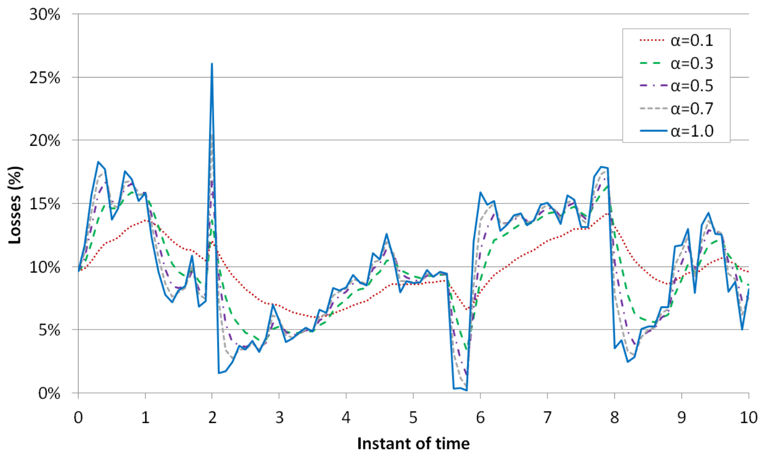


Fig. 5.24. Average losses distribution and smooth effect in scenario 4.

The methodology used is the same employed in the previous studies, that is, in each instant of time clients report their losses, the server smoothes them and then it chooses the optimum code rate according to this information. It should be noted that clients only inform about the losses they are perceiving in a specific instant of time. For example, in the instant of time 1, clients only inform about the losses they perceive in that instant, and not about the partial losses perceived between the instants of time 0 and 1.

Fig. 5.25 compares the average download time obtained for different values of α for each instant of time. In the figure, the download time is in range [8000 ms, 12 000 ms]. Due to the smooth process, in those cases where the value of α is low, the server tends to be in the same protection state. Fig. 5.25 shows that, in this scenario, smoothing the losses provides better results. Specifically, the system adapts better to the changing losses using $\alpha = 0.3$ or $\alpha = 0.5$. Analyzing the system behavior for $\alpha = 0.3$, we can see that the server sends at an optimum code rate in 8 out of 10 time intervals. This is a very good result, since in the instants of time when the server does not send with the optimum code rate, the difference between the adaptive and the optimum code rate is pretty minimal. Therefore, we can consider that the adaptive system performs very well using an appropriate smooth factor. Obviously, an erroneous estimation of the losses can increase the average download time of the clients as well as the bandwidth.

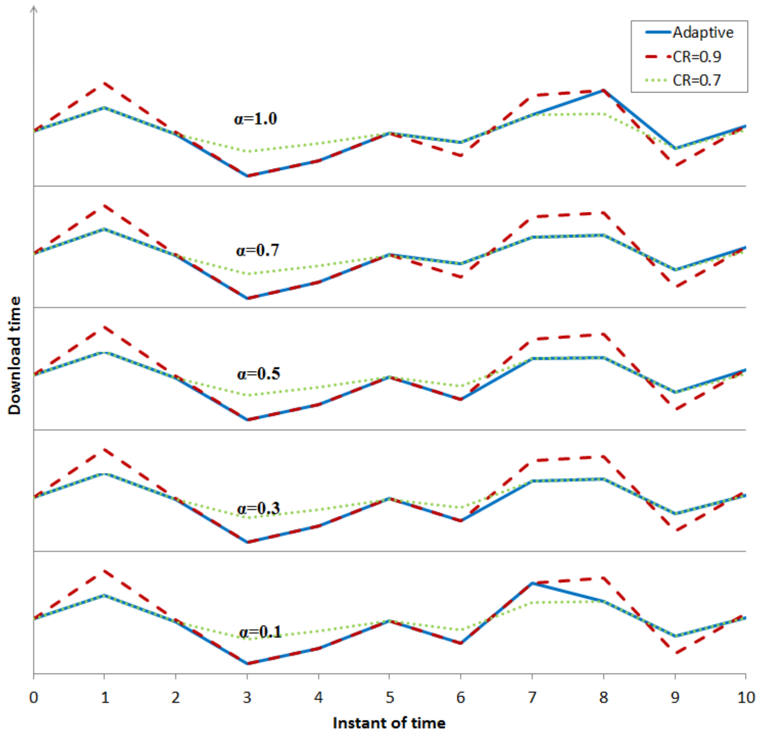


Fig. 5.25. Download time evaluation for different values of α in scenario 4 for N_h = 1/6, N_m = 1/3.

5.4.6. Delivery frequency of the feedback messages

Regarding the transmission of the feedback messages from the clients to the server, there are two main methods to send these reports: synchronously or asynchronously. In the first, clients send periodically their losses reports with a delivery frequency which depends on, for instance, the number of users and the available bandwidth. In the latter, clients inform about their losses only when it is necessary (for instance, when they detect a meaningful change of losses). As an example, it is worth mentioning that RTP/RTCP [78] implement both mechanisms: a synchronous one, where users send their reports with a certain offset to avoid the feedback implosion problem; and an asynchronous mechanism (immediate feedback), where users report the server only when required.

In this way, when choosing the best policy to decide the delivery period of the feedback messages, there are several things to take into account:

- If the delivery period of the feedback messages is very short, too much traffic is introduced into the network. There would be many collisions between the feedback messages of all clients, which could lead to network congestion. Also, if losses do not change excessively it is not worth sending too many reports.
- Moreover it is not optimal to change continuously the code rate: clients have to discard continuously the parity packets previously received and generate the parity matrix. If clients do not send very frequently their reports, the server does not need to recalculate and generate a new optimum code rate constantly.
- On the other hand, sending feedback messages with a low frequency could become the system inefficient.

In this sense, we consider a good alternative that clients send the feedback messages each time they receive a FLUTE block. It should be remembered that in FLUTE the coding is generated by block, so different blocks of the same file can have different coding and/or code rate. Thus, the delivery period of the feedback messages should be equal or higher than the transmission time of a FLUTE block. This solution is similar to the one used by DASH clients to request for new segments each time a segment is received.

The size of a block depends on the parameters established in the blocking algorithm (such as the code rate or the encoding symbol length). As explained, LDPC codes work more efficiently with higher block sizes [51]. But when choosing the size of the blocks it should be taken into account the computational resources of the mobile receivers (the higher the block size the more complex the decoding), so there is a trade-off between coding efficiency and computational cost.

As we have seen in previous chapters, a block size of 3000 FLUTE packets offers good results of the coding efficiency and the decoding time. Precisely, the experiments presented in this chapter have been carried out with blocks of 3000-packet size. According to the results presented in Table 5.1, the transmission of a block of this size takes among 7000 and 15 000 ms using the optimal code rate for different channel losses. Thus, assuming an average of 10 000 ms, with an average feedback delivery period of 10 000 ms and 100 users in the system, the bandwidth consumed by the feedback messages will be around 8 kbps (considering that the feedback packet size is around 100 B). This is a good value taking into account that the multicast transmission rate considered is 5 Mb/s. Also, following the DASH example, a feedback time of 10 s is reasonable compared to the feedback times used to request DASH segments (2 s in Microsoft Smooth Streaming or 10 s in Adobe HTTP Dynamic Streaming [86]).

These results show that the proposal of sending feedback messages after receiving each block results convenient in terms of bandwidth, which is one of the premises of this chapter.

5.4.7. Evaluation of the hysteresis

The last study evaluates how the hysteresis affects two main parameters: the average download time and the number of times that the state protection changes. If the server changes their protection state too frequently, the adaptive system could become inefficient: the server is changing the code rate continuously whereas the clients are consuming more computational resources since they have to process a lot of coding changes. Thus, there is a trade-off between minimizing the download time and the number of state changes.

Scenario 5 is used to evaluate the hysteresis, shown in Fig. 5.11. In that scenario, the average losses for each instant of time fluctuates in range [8%, 17%], in this way the state protection changes very frequently. With the aim of evaluating the hysteresis effect, we fix the lower threshold (λ_L) to 7% and we increase the upper threshold (λ_M) progressively. Fig. 5.26 shows the results obtained regarding the number of state changes and the average download time of all clients considering all the time intervals. In the graph, the *x* label represents the threshold interval, that is, λ_M - λ_L. Thus, a value of threshold interval equal to 1 indicates that: λ_L = 7% and λ_M = 8%. It should be noted that, in this study we establish N_h = 1/3 and N_m = 2/3, so that the high losses do not mask the effect of the hysteresis.

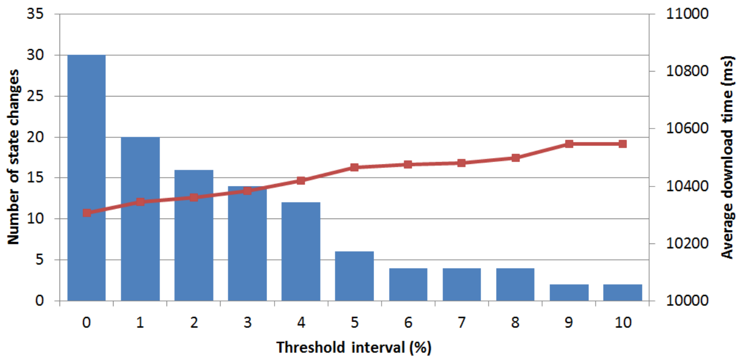


Fig. 5.26. Number of state changes and average download time depending on the threshold interval in scenario 5 for N_h = 1/3, N_m = 2/3.

Obviously, as the threshold interval increases, the number of state changes is lower, since the system tends to be in the same state protection. However, as the threshold interval is higher the average download time increases.

As Fig. 5.26 shows, the threshold interval used in the previous studies (5%, $\lambda_L = 7\%$ and $\lambda_M = 12\%$) represents a good trade-off, since there are not very much changes of protection state and the average download time does not get worse considerably regarding the minimum value (only 1.5% higher).

5.4.8. Server algorithm

After evaluating all the parameters presented in the chapter, this last subsection presents an algorithm (Algorithm 5.3) that summarizes the process carried out by the server to obtain the best code rate for each instant of time so as to provide an optimal delivery.

It is worth mentioning that the algorithm is heuristic, since it uses the different parameters obtained throughout the evaluation section. The fact that the algorithm works well for the five scenarios evaluated provides a certain guarantee to be valid in other scenarios.

Algorithm 5.3: Server procedure

```
INPUT: feedbacks from clients informing about the losses perceived for each instant of
        time
OUTPUT: new code rate (CR)
1:  Fix parameters ($\lambda_L = 7\%$, $\lambda_M = 12\%$, $\lambda_H = 30\%$)
2:  Initialize ($CR = 0.9$, $N_h = 1/6$ and $N_m = 1/3$, $\alpha = 1$)
3:  for (each instant of time)
4:      Receive feedback from clients, generate "losses_vector" and calculate $n_H$, n
5:      if ($n_H/n \leq 5\%$) then $N_h = 1/3$, $N_m = 2/3$
6:      else then $N_h = 1/6$, $N_m = 1/3$
7:      end
8:      if (losses variation $\geq 10\%$) then $\alpha_{current} = 0.8 * \alpha_{prev}$
9:      else then $\alpha_{current} = \alpha_{prev} + 0.1$
10:     end
11:     Calculate "smooth_losses_vector" and $n_M$, $n_H$, n
12:     if ($n_H/n \geq N_h$) then CR = 0.7
13:     else if (($n_M + n_H)/n \geq N_m$) then CR = 0.7
14:     else then CR = 0.9
15:     end
16: end
```

Initially, the server fixes a conservative value of the code rate (CR = 0.9) in order to save bandwidth. Also, the server considers an environment of low client mobility (so $\alpha = 1$) and that clients with high losses have more priority than those with low losses ($N_h = 1/6$ and $N_m = 1/3$). According to the previous studies, the parameters that delimit each losses region are fixed to $\lambda_L = 7\%$, $\lambda_M = 12\%$ and $\lambda_H = 30\%$.

For each instant of time the server creates a vector with the losses of those clients who have sent losses reports. If the percentage of clients with high losses is very low (≤ 5%), the server could give the same priority to all losses regions by assigning $N_h = 1/3$ and $N_m = 2/3$. Next, the server checks if the percentage of losses perceived by each client has changed a lot regarding the previous instant of time, by comparing the current losses vector with the previous. If the average increment or decrement of losses perceived by the clients is higher than a 10%, the server increases the value of α. Otherwise the server reduces α (note that $0 \leq \alpha \leq 1$). We apply the additive-increase/multiplicative-decrease algorithm (using the results obtained in the studies carried out in this chapter), which is used by RTP/RTCP for dynamic adjustment of the bandwidth as well as TCP (Transmission Control Protocol) to manage network congestion. Then the server calculates the smoothed losses of all clients, generating a smooth losses vector. After that, the server counts the number of clients in each losses region and calculates the new protection state, and thus the code rate.

5.5. Conclusions

The adaptive system presented in this chapter represents a good solution for file transmission in multicast environments with limited bandwidth. The use of an adaptive code rate minimizes the average download time of all clients within the coverage area, with a reasonable use of bandwidth.

Although there is an optimum code rate per each client depending on the amount of losses perceived, a certain code rate provides good values of download time in a wide range of packet losses around the code rate that minimizes the download time. Therefore, it is possible to send using a code rate that benefits the major part of users. In order to do this, it is necessary to analyze the losses perceived by each client and decide the optimum code rate for each situation. In the studies carried out two different code rates have been considered: one code rate when the major part of clients have low losses and another one when they have medium-high losses. Clients with high losses must have more priority than those with low losses, since using insufficient protection for clients with high losses penalizes more the download time than using too much protection for clients with low losses. As the results have shown, the adaptive system proposed works very well using only two different code rates. The value of these code rates has a great impact on the system performance, as well as the thresholds that delimit the protection state of the system, which establish the code rate used to transmit. In this sense, it is recommended the use of hysteresis to avoid too much coding changes.

As a particular case, this chapter has shown a carousel where the server sends the same file in each loop and clients download the file in one or several carousel cycles, depending on the losses. In that case, the adaptive system performs rather well, despite the fact that the optimum code rate could change every carousel cycle and clients must discard parity packets previously received.

Finally, in environments where the losses perceived by the users change very abruptly, it is recommended that the server smoothes the losses when it chooses the optimum code rate. In that case, choosing an accurate smooth factor has a great influence on the suitable performance of the adaptive system.

To sum up, the adaptive mechanism proposed in this chapter represents a good trade-off between the bandwidth used by the file server and the Quality of Experience perceived by the clients, therefore it is appropriate for content download services in multicast wireless networks. Comparing to the results of the previous chapter, the system proposed in this chapter reduces the bandwidth at the expense of increasing the download time.

This chapter has led to the following work in an international journal: [J.7].

Chapter 6

Analysis and evaluation of the File Delivery Table

The key element of the FLUTE protocol is the use of the File Delivery Table (FDT), which is the in-band mechanism used by FLUTE to inform clients about the files (and their characteristics) transmitted within a FLUTE session. Clients need to receive the FDT in order to start downloading files. Thus, the delivery of FDT packets and the proper configuration of their parameters have a great impact on the Quality of Experience perceived by the users of FLUTE content download services. This chapter presents a complete analysis about how the FDT transmission frequency affects the download time of files. Moreover, results show which are the optimum values that minimize this download time. An appropriate configuration of the FDT transmission frequency as well as the use of AL-FEC mechanisms provides an optimum content delivery using the FLUTE protocol.

6.1. Introduction

As seen throughout this thesis dissertation, FLUTE is the standard protocol used in unidirectional environments to provide reliability in the transmission of multimedia files. The main characteristic of FLUTE is the use of an in-band signaling mechanism, i.e. the File Delivery Table. The FDT describes the main properties of the files that are being transmitted, such as the file name, the content length or the encoding type. Once the clients receive the FDT, they know the files the server is delivering and can start downloading them.

On the other hand, as commented in the introductory section of this work, the Quality of Experience perceived by the users is an important parameter in the evaluation of both video streaming and file transmission services. In file transmissions, a good QoE is obtained when the file is received correctly with the minimum download time. This is accomplished by sending files with a high transmission rate in channels with minimum losses. But these conditions cannot always be controlled and therefore it is essential to send the content in the most efficient way.

Regarding this, file transmissions through FLUTE can be optimized to improve the efficiency of the content delivery. Previous chapters explained how AL-FEC mechanisms help to improve the file transmission. Also, the FDT is a key element in FLUTE sessions and how often the FDT is transmitted has an important impact on the QoE of FLUTE services. It should be remembered that first clients have to receive the FDT and then they are able to start downloading files. In this way, if the FDT is sent with low frequency, clients have to wait a long time until they can start downloading the contents. On the other hand, if the FDT is received too frequently, clients have to wait less time to receive the FDT (and therefore to start the download) at the expense of receiving more useless packets afterwards, thus reducing the efficiency of the transmission.

In this sense, this chapter analyzes how the transmission of the FDT affects FLUTE file delivery sessions and evaluates which are the optimum values of the FDT transmission frequency that minimize the download time and, therefore, improve the Quality of Experience perceived by the users.

In Section 2.4 the related work of FLUTE was explained. It is worth highlighting that, despite that the FDT is one of the most characteristic elements of FLUTE, there are no papers where the FDT is analyzed in detail. Also, neither the original FLUTE RFC [7] nor the new FLUTE standard [2] analyze how often an FDT Instance should be sent and how much FEC protection should be provided for each FDT Instance. This chapter is intended to fill this gap.

6.2. Theoretical analysis

6.2.1. Introduction

The objective of this study is to calculate the total download time (t_T) of a file and evaluate how it is affected by the FDT delivery configuration. Throughout this section several variables appear. Table 6.1 shows these variables and explains their meaning.

Table 6.1. Key notation.

b	Transmission rate	n_L	Number of packets that make up the file to download
CR	Code rate	S_i	Size of file *i*
CR_f	Code rate applied to files	S_L	Size of the file to download
CR_{FDT}	Code rate applied to FDT Instances	t_C	Cycle time
D_{FDT}	Number of data packets sent between two FDT Instances	t_D	Download time
e	Encoding symbol length of a FLUTE data packet	t_{FDT}	Delivery period of FDT Instances
e'	Encoding symbol length of a FLUTE FDT Instance packet	t_M	Time passed since the FDT is received until the client receives the first packet of the file to download
m	Number of times per cycle that an FDT is sent	t_S	Time needed to send once the file to download
m_L	Number of FDT Instances of file *L* sent in each carousel cycle	t_T	Total download time
N	Number of files in the carousel	t_W	Waiting time
n_{CR}	Number of packets that make up a file after applying AL-FEC	α	Adjustment factor of the file size
n_{cycles}	Number of cycles needed to download a file	α_L	Adjustment factor of the file size referred to the file *L*
n'_{cycles}	Entire number of the n_{cycles}, defined as: ceil (n_{cycles})	β	In the last cycle, it represents the remaining cycle percentage to complete an entire cycle, referred to the entire carousel
$n_{cyclesFDT}$	Number of cycles needed to download an FDT	β_L	In the last cycle, it represents the remaining cycle percentage to complete an entire cycle, referred to the file to download
n_i	Number of packets that make up file *i*	γ	Number of packets that compose an FDT

This section analyzes five transmission configurations, each one explained in a different subsection as follows. Section 6.2.2 presents the general case: download of one file, use of a complete FDT and sequential scheduling. Section 6.2.3 analyzes partial FDTs, downloading one file and using sequential scheduling. Section 6.2.4 studies the scheduling model: download of one file, use of a complete FDT and interleaving scheduling. Section 6.2.5 considers multiple downloads, using a complete FDT and sequential scheduling. Finally, Section 6.2.6 analyzes the use of prefetching with the parameters of the general case: download of one file, use of a complete FDT and sequential scheduling.

6.2.2. General case

As explained in Chapter 2, the RFC of FLUTE establishes that there are two types of FDT Instances: partial FDT and complete FDT, since each FDT Instance contains at least a single file description entry and at most the complete FDT of the file delivery session. As a first study case, among the two types of FDTs, this section analyzes the complete FDT. Each FDT Instance is transmitted in γ FLUTE packets, depending on the number of files and attributes that the FDT Instance describes. This study evaluates different cases: sending only one FDT Instance per cycle; sending as many FDT Instances as files in the carousel; and sending FDT Instances more frequently.

In this study, it is assumed that a server sends N files sequentially in a FLUTE carousel and that a user wants to download a certain file L with size S_L. Also, the server sends files always using the same coding and code rate, so the carousel size is equal for every cycle. Fig. 6.1 shows an example of a FLUTE delivery session based on carousels where three files are sent. It should be noted that the FDT Instances are sent in-band with the files. In the example, a certain client wants to download the file F3.

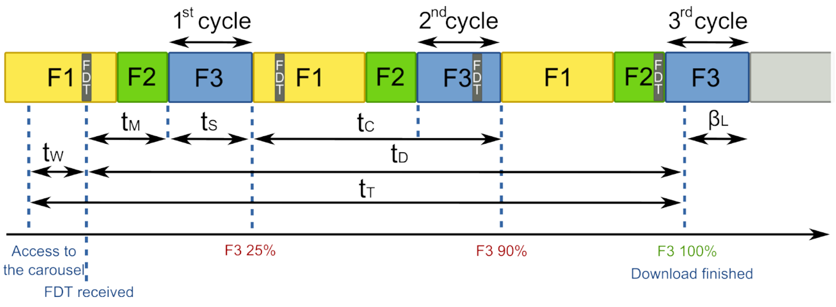


Fig. 6.1. Example of a carousel transmission.

Assuming that clients connect to the channel after the server starts sending content, a client will access to the carousel in a certain moment and will not be able to download the file until that client receives the FDT. So, the client needs to wait a certain time to receive the FDT, called waiting time (t_W). After receiving the FDT, the client needs a time to download the file (t_D). In this way, the total download time needed to complete a download (t_T) is

$$t_T = t_W + t_D. \qquad (6.1)$$

Regarding the calculation of the download time (t_D), this is composed by three different times, as Fig. 6.1 shows. Firstly, there is a time t_M that indicates the time passed since the FDT is received until the client receives the first packet of the file to download. In that example, when the client accesses to the carousel, F1 is being transmitted. Therefore, the client will have to wait until the server finishes the transmission of F1 and then the server sends F2 completely. Secondly, t_S indicates the time needed to send once the file to download, that is, the time passed from the transmission of the first packet until the transmission of the last one. As Fig. 6.1 shows, it is very likely that, if there are losses during the transmission, the client needs more than one carousel cycle to download a certain file. In the example, after the first cycle, the client has downloaded a 25% of the file, whereas after the second cycle, the client has downloaded 90%. The download is completed during the third cycle. Thus, the download time also depends on the number of cycles needed to complete the download. Therefore, the download time is calculated as

$$t_D = t_M + t_S + t_C \cdot (n_{cycles} - 1). \tag{6.2}$$

Hence, the cycle time (t_C) has a great impact on the download time. As we have seen in previous chapters, the use of AL-FEC mechanisms allows to reduce the number of carousel cycles. In the case of considering AL-FEC, the code rate must be taken into account when calculating the cycle time:

$$t_C = \frac{\sum_{i=1}^{N} ceil\left(\frac{S_i}{CR_f}\right) + ceil\left(\frac{m \cdot e' \cdot \gamma}{CR_{FDT}}\right)}{b}, \tag{6.3}$$

where N is the number of files in the carousel; S_i is the size of the file i; m is the number of FDT Instances sent during a carousel cycle; e' is the size of an FDT Instance packet; γ is, as mentioned, the number of packets that compose an FDT Instance; CR_f and CR_{FDT} represent the code rate used to send the files and the FDT Instances, respectively; and b is the transmission rate. It should be noted that S_i can be calculated as the product of the number of packets of the file i (n_i) and the size of FLUTE packets (e). Also, the use of AL-FEC encoding generates more packets. The total number of packets after coding is calculated by ceiling the division of the number of packets by the code rate. Assuming that both the files and the FDT will have the same protection (and thus the same code rate) and not considering the effect of the ceiling round, formula (6.3) is simplified:

$$t_C \cong \frac{\sum_{i=1}^{N} S_i + m \cdot e' \cdot \gamma}{b \cdot CR}. \tag{6.4}$$

It must be stressed that this formula and the following ones are valid when No-FEC is used, simply assigning $CR = 1$.

Returning to (6.1), with regard to the waiting time (t_W), this is a random variable with expected value $E[t_W]$. In this study, it is assumed that the instant of time when the client accesses the carousel is uniformly distributed in the interval [t, $t + t_{FDT}$], where t_{FDT} is the delivery period of the FDT Instances, that is:

$$t_{FDT} = \frac{t_C}{m}. \tag{6.5}$$

In this way, the expected waiting time is

$$E[t_W] = \frac{t_C}{2m}. \tag{6.6}$$

The delivery period of FDT Instances is directly related to the parameter D_{FDT}, which indicates how many packets are sent between two FDT Instances:

$$D_{FDT} = \frac{\sum_{i=1}^{N} n_i + m \cdot \gamma}{m} \cong \frac{t_{FDT} \cdot e}{b}. \tag{6.7}$$

The approximation of the latter expression is due to the fact that the encoding symbol length of the data packets (e) and the FDT Instances (e') can be different.

Moreover, if there are losses in the transmission, it must be taken into account that FDT Instances can be lost, so several FDT cycles ($n_{cyclesFDT}$) –considering that an FDT cycle is the period between the delivery of two FDT Instances (t_{FDT}) – can be necessary to obtain the FDT. Thus, formula (6.6) becomes

$$E[t_W] = \frac{t_C}{2m} + \frac{t_C}{m} . (n_{cyclesFDT} - 1). \tag{6.8}$$

Simplifying this expression yields

$$E[t_W] = \frac{t_C}{m} . \left(n_{cyclesFDT} - \frac{1}{2} \right). \tag{6.9}$$

As (6.9) shows, the expected value of the waiting time decreases as the number of FDT Instances sent in the carousel is higher and as the number of cycles is lower.

In order to calculate this number of cycles, the methodology explained in [45] and in Section 4.3 can be used. The first considers uniformly distributed losses whereas the latter models the channel losses using the Markov model. Again, as considered throughout this thesis work, this chapter considers bursty losses, thus the Markov model [57] is used. Therefore, in order to calculate the number of cycles, we consider the same methodology used in Chapter 4 and Chapter 5. In this way, we use again equations (4.1)-(4.3) to calculate the number of cycles when No-FEC is used, and equations (4.4)-(4.6) when AL-FEC is used.

On the other hand, in regard to the calculation of the expected download time (t_D), as explained before this is composed by three terms: t_M, t_S and the number of cycles needed to complete the download.

$$E[t_D] = E[t_M] + E[t_S] + E[t_C \cdot (n_{cycles} - 1)] \tag{6.10}$$

Concerning t_M, it is possible to obtain an analytical expression considering m possible cases, each one corresponding to an FDT Instance in the file carousel. It is assumed that the client accesses the carousel at any instant $[t, t + t_C]$ and therefore, the m possible cases have equal probabilities, yielding to the following approximation:

$$E[t_M^L] \approx \frac{t_C}{2} - \frac{S'_L}{2b}, \tag{6.11}$$

where S'_L is the size of the file to download adjusted by the number of FDT Instances sent during the delivery of that file per carousel cycle. That adjustment is represented by the variable α. Also, S'_L is adjusted by the additional packets transmitted because of the use of AL-FEC. Thus,

$$S'_L = \frac{S_L \cdot \alpha}{CR}, \tag{6.12}$$

$$\alpha = 1 + \frac{m \cdot \gamma}{\sum_{i=1}^{N} n_i}. \tag{6.13}$$

Returning to (6.10), the calculation of t_S is more intuitive:

$$E[t_S^L] = \frac{S'_L}{b}. \tag{6.14}$$

On the other hand, in (6.10) the number of cycles is calculated using again formulas (4.1)-(4.6). This number of cycles has a great impact on the t_T and, as mentioned, is directly related to the losses in the transmission channel. It is worth noting that the client, in order to complete the download, needs actually an entire number of cycles (n'_{cycles}) minus a percentage of the last cycle (β_L), where β_L represents the download percentage referred to the file. In this sense, β is defined as the download percentage of the entire carousel. In the example of Fig. 6.1, the client needs 3 cycles to complete the download minus a little percentage of the last cycle.

$$n_{cycles} = n'_{cycles} - \beta. \tag{6.15}$$

Algorithm 6.1 and Algorithm 6.2 show how to calculate both n'_{cycles} and β, for the case of No-FEC and AL-FEC, respectively. Both algorithms use the formulas presented in (4.1)-(4.6), with the following input parameters: the number of packets that make up a file (n), the percentage of channel losses and the maximum number of iterations used to calculate β. Also, Algorithm 6.2 receives as input parameters the number of packets of the file after decoding (n_{CR}) and the inefficiency ratio. Each algorithm calculates the number of new packets received in each cycle and checks if enough packets have been received to rebuild the file. In that case, it adjusts the percentage of the last cycle that completes the download. This is calculated through an iterative process with a precision determined by the maximum number of iterations. For instance, with 10 iterations there is a precision smaller than 10^{-3} cycles.

Algorithm 6.1: Calculation of the download time (No-FEC)

```
INPUT: n, losses, max_iter
OUTPUT: $n'_{cycles}$, $\beta_L$
1:  Initialize ($n'_{cycles} = 1$, $\beta_L = 0$, packets_received = 0)
2:  Calculate number of losses per cycle (l = n * losses) and number of packets
    received at first loop
3:  while (not all packets have been received)
4:      $n'_{cycles} = n'_{cycles} + 1$;
5:      Calculate new packets received (P) in the current loop using (4.2)
6:      if (packets_received + P = n)
7:          Initialize (bottom = 0, top = 1, thres = 0, iter = 1)
8:          while (iter < max_iter)
9:              thres = (bottom + top)/2
10:             Calculate new packets received (P) with (4.2) and input parameters:
                thres * (n, packets_received, l)
11:             if (packets_received + P = n)
12:                 top = thres;
13:             else
14:                 bottom = thres;
15:             endif
16:             iter = iter + 1;
17:         endwhile
18:         $\beta_L = 1 - \text{thres}$;
19:         BREAK
20:     else
21:         packets_received = packets_received + P;
22:     endif
23: endwhile
```

Algorithm 6.2: Calculation of the download time (AL-FEC)

```
INPUT: n, $n_{CR}$, losses, inef_ratio, max_iter
OUTPUT: $n'_{cycles}$, $\beta_L$
1:  Initialize ($n'_{cycles} = 1$, $\beta_L = 0$, packets_received = 0)
2:  Calculate number of losses per cycle (l = $n_{CR}$ * losses) and number of packets
    received at first loop
3:  while (not enough packets have been received)
4:      $n'_{cycles} = n'_{cycles} + 1$;
```

```
5:      Calculate new packets received (P) in the current loop using (4.5)
6:      **if** (packets_received + P $\geq$ n * inef_ratio)
7:          Initialize (bottom = 0, top = 1, thres = 0, iter = 1)
8:          **while** (iter < max_iter)
9:              thres = (bottom + top)/2
10:             Calculate new packets received (P) with (4.5) and input parameters:
                thres * ($n_{CR}$, packets_received, l)
11:             **if** (packets_received + P $\geq$ n * inef_ratio)
12:                 top = thres;
13:             **else**
14:                 bottom = thres;
15:             **endif**
16:             iter = iter + 1;
17:         **endwhile**
18:         $\beta_L$ = 1 – thres;
19:         **BREAK**
20:     **else**
21:         packets_received = packets_received + P;
22:     **endif**
23: **endwhile**
```

Both algorithms return the entire number of cycles (n'_{cycles}) and the percentage of the last cycle that completes the download referred to the file to download (β_L). In order to calculate the β referred to the entire carousel, formula (6.16) is used:

$$\beta = \beta_L \cdot \frac{S'_L \cdot CR}{\sum_{i=1}^{N} S_i + m \cdot e' \cdot \gamma}. \tag{6.16}$$

In this way, taking into consideration expressions (6.11)-(6.16), the expected value of t_D of a file L is obtained by replacing in (6.10):

$$E\left[t_D^L\right] \approx \frac{t_C}{2} - \frac{S'_L}{2b} + \frac{S'_L}{b} + t_C \cdot \left(n'_{cycles} - \beta - 1\right). \tag{6.17}$$

That expression can be simplified:

$$E\left[t_D^L\right] \approx t_C \cdot \left(n'_{cycles} - \beta - \frac{1}{2}\right) + \frac{S_L \cdot \alpha}{2b \cdot CR}. \tag{6.18}$$

Finally, once the expected waiting time and the expected download time are calculated, the estimated value of the expected total download time of a file L is calculated replacing in (6.1):

$$E\left[t_T^L\right] \approx t_C \cdot \left(\frac{n_{cyclesFDT}}{m} + n'_{cycles} - \beta - \frac{1}{2} - \frac{1}{2m} \right) + \frac{S_L \cdot \alpha}{2b \cdot CR}. \tag{6.19}$$

For the sake of clarity, we sum up the main formulas presented. The expected download time of a certain file is calculated using (6.19), where t_C is obtained with formula (6.3) or (6.4), α is calculated with (6.11), and the number of cycles (both the file and the FDT) is obtained by means of Algorithm 6.1 (if No-FEC is used) or by means of Algorithm 6.2 (if AL-FEC is used).

It should be noted that the procedure is similar to the one explained in Section 5.3.1 when calculating the download time. The differences regarding Section 5.3.1 are: firstly, this section considers the waiting time; then, in this section the server uses the same coding and code rate during the transmission; and finally, Section 5.3.1 did not consider the effect of FDT Instances when calculating the download time.

6.2.3. Partial FDT

The theoretical analysis presented in previous section assumes that the FDT sent by the server is complete, that is, it describes all files of the carousel. The calculation of the expected total download time of a file with partial FDT is very similar. The main difference regarding (6.19) is the value of m, which must be replaced by a partial m (m_L), which indicates how many FDT Instances describing file L are sent in each carousel cycle.

$$E\left[t_T^L\right] \approx t_C \cdot \left(\frac{n_{cyclesFDT}}{m_L} + n'_{cycles} - \beta - \frac{1}{2} - \frac{1}{2m_L} \right) + \frac{S_L \cdot \alpha_L}{2b \cdot CR}. \tag{6.20}$$

Supposing that each partial FDT only carries information of one certain file, the sum of all m_i must be equal to m:

$$m = \sum_{i=1}^{N} m_i. \tag{6.21}$$

Moreover, the value of α_L could be different depending on the way partial FDT Instances are sent. If they are sent throughout the carousel, α_L will be equal to expression (6.13), whereas if partial FDT Instances are only sent during the transmission of the file they describe, then α_L will be calculated using the next expression:

$$\alpha_L = 1 + \frac{m_L \cdot \gamma}{n_L}. \tag{6.22}$$

In partial FDT the number of cycles needed to rebuild an FDT ($n_{cyclesFDT}$) will be lower or equal to the ones needed in the previous case since FDT Instances will be made up by less packets (in general, $\gamma = 1$ if FDT Instances only describe one file).

6.2.4. Interleaving

On the other hand, server may interleave the packets belonging to different files, instead of using a sequential transmission. The waiting time is not affected by the transmission model (only by the transmission frequency of FDT Instances and the cycle time), so formula (6.9) remains valid. Nevertheless, when calculating the download time – formula (6.10) – both t_M and t_S change. The term referred to the number of cycles is equal. Specifically, as packets are interleaved, the time passed since the FDT is received until the first packet of the file to download is received, that is t_M, is reduced. When calculating t_M, considering that packets are interleaved according to a periodic sequence, it is taken into account that files with higher sizes will have a lower t_M. This time is calculated by multiplying the time needed to transmit a packet by the size of the whole carousel and divided by the size of the file to download. Thus, the expected t_M of a file L is

$$E\left[t_M^L\right] = \frac{e}{b} \cdot \frac{\sum_{i=1}^{N} S_i + m \cdot e' \cdot \gamma}{2S'_L \cdot CR} = \frac{t_C}{2} \cdot \frac{e}{S'_L}. \tag{6.23}$$

On the contrary, the value of t_S increases considerably. This value is different depending on whether FEC is used or not. Without FEC, on average, the client must wait until almost the end of the carousel to get the last packet of the file to download. Thus, following a similar reasoning that in formula (6.22), it is possible to calculate the expected value of t_S:

$$E\left[t_S^L\right] = t_C \cdot \frac{\frac{S'_L}{e} - 1}{\frac{S'_L}{e}}. \tag{6.24}$$

If AL-FEC is used, the value of t_S is in the range

$$E\left[t_S^L\right] = \left[t_C \cdot \frac{e}{S'_L}(n * inef\_ratio - 1), t_C \cdot \frac{e}{S'_L}(n_{CR} - 1) \right]. \tag{6.25}$$

The best case is when there are no losses, so the client has to receive n * *inef_ratio* packets to rebuild the file. As the losses increase, t_S increases too, with a maximum value of almost t_C. Thus, the total download time is obtained as in the general case, that is, considering t_W, t_M, t_S and the number of cycles necessary to complete the download. In any case, the download time using interleaved scheduling will always be higher than the one obtained using sequential transmission. Nevertheless, the use of interleaving could be highly recommended in order to minimize the effect of burst losses.

6.2.5. Multiple download

Back to the case of sequential scheduling, if the client decides to download all files of the carousel instead of only one, the total download time is calculated in a similar way. In this case, the waiting time would be the same that in the general case, calculated using formula (6.9). On the other hand, the term t_M disappears. In this case, the download time is determined by the file that takes longer to download. In general, this file will be the largest file in the carousel, because its download requires more carousel cycles. The expression of the total download time of the entire carousel is

$$E[t_T] \approx \frac{t_C}{m} \cdot \left(n_{cyclesFDT} - \frac{1}{2} \right) + t_C \cdot (n'_{cycles} - \beta). \tag{6.26}$$

6.2.6. Prefetching

In this regard, it would be interesting to start storing packets before the FDT is received, in order to reduce the total download time. In this sense, the RFC of FLUTE [2] indicates that, although generally a receiver needs to receive an FDT Instance describing a file before it is able to recover the file itself, when packets are received before the FDT, the system performance might be improved by caching such packets within a reasonable time window and storage size. Therefore, the client can save packets although the client does not know which file the packets belong to until the client receives the FDT. This can be useful in environments where the size of the data carousel is not very high and the client does not have limited resources or storage problems. In this case, the total download time can be reduced considerably. Regarding the waiting time, this will be the same that in the previous analysis, that is:

$$E[t_W] = \frac{t_C}{m} \cdot \left(n_{cyclesFDT} - \frac{1}{2} \right). \tag{6.27}$$

If the client starts saving data before the FDT is received, the expression of the download time does not change, considering that, in this situation, t_M represents the time passed since the client connects to the channel until the client receives the first packet of the file to download (although the client is storing all the packets that it receives), so the formula that defines the download time is

$$E[t_D^L] \approx t_C \cdot \left(n'_{cycles} - \beta - \frac{1}{2} \right) + \frac{S_L \cdot \alpha}{2b \cdot CR}. \tag{6.28}$$

As clients need to receive the FDT to assign the properties to the file, the total download time will be specified by the waiting time if all packets are received before the FDT does, or by the download time otherwise:

$$E[t_T^L] \approx \max\left\{ \frac{t_C}{m} \cdot \left(n_{cyclesFDT} - \frac{1}{2} \right), t_C \cdot \left(n'_{cycles} - \beta - \frac{1}{2} \right) + \frac{S_L \cdot \alpha}{2b \cdot CR} \right\}. \tag{6.29}$$

Finally, it should be recalled that all presented formulas remain valid when No-FEC is used, by fixing the code rate to 1.

6.3. Evaluation methodology

This section presents the evaluation of the FDT transmission frequency, first for the general case (Section 6.4.1) and then for some particular cases. Specifically, Section 6.4.2 analyzes the interleaving scheduling and Section 6.4.3 evaluates the advantages of using prefetching.

The main evaluation parameter is the total download time (t_T). This section shows various results for different values of the number of files in the carousel (N), different AL-FEC protection, and for different sizes and file distributions of packets (S_i).

The evaluation of the FDT transmission frequency has been done through different simulations, each comprised of as many iterations as needed to provide narrow (99%) confidence intervals. The studies consider different values of the number of times an FDT Instance is sent in each carousel cycle (that is, different values of m, which imply different values of D_{FDT}). Specifically, the values of m considered have been: 1, $N/50$, $N/10$, $N/5$, $N/2$, N, $2N$, $5N$, $10N$, and $50N$. These values of m provide the following approximated values of D_{FDT}: n_i*N, n_i*50, n_i*10, n_i*5, n_i*2, n_i, $n_i/2$, $n_i/5$, $n_i/10$, and $n_i/50$ packets respectively.

As in previous chapters, in the evaluation, a packet size (e) of 1428 bytes has been used, according to the maximum transfer unit (MTU) of Ethernet (1500 bytes) and the results presented in [45]. With that size, and with typical file parameters in an FDT (such as content location, TOI or file size), it is assumed that one packet can contain over six file descriptions. In this way, $\gamma = \text{ceil}(N/6)$. Also, it is supposed that $e' = e$. Moreover, a transmission rate (b) of 5 Mb/s has been used, although the value of this parameter does not affect the conclusions of this study. Regarding channel losses, the different studies consider errors from 0 to 50% packet losses, in steps of 5%.

In relation to AL-FEC, it is considered that LDPC codes are used. Nevertheless, the studies hereby presented remain valid independently of the codes used. The only difference in the results presented is the value of the inefficiency ratio. In this case, it is considered a value of inefficiency ratio equal to 1.07, according to [51] and [77], as we considered in Chapter 4 and Chapter 5.

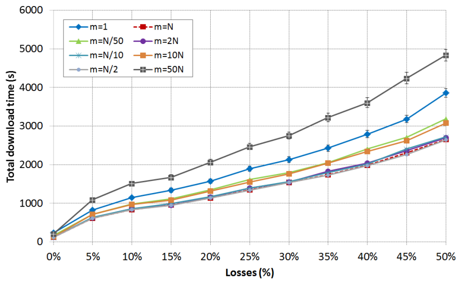
Finally, it is worth mentioning that in the studies the file L to download will always be the largest of the carousel.

6.4. Results and analysis

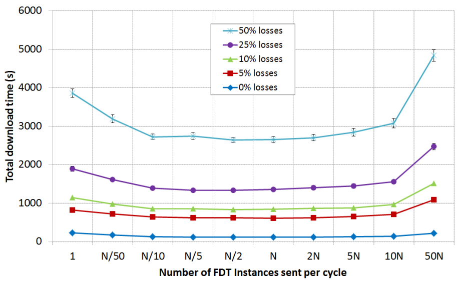
6.4.1. General case

As a first study case, it is considered that a server sends a data carousel with $N = 100$ files of $n_i = 1000$ packets (that is, $S_i = 1\,428\,000$ bytes). These values of N and n_i, according to (6.7), provide the following values of D_{FDT}: 100 000, 50 000, 10 000, 5000, 2000, 1000, 500, 200, 100, and 20 packets.

Fig. 6.2(a-b) show the results of the total download obtained when No-FEC is used for different configurations of the FDT frequency and channel losses. Fig. 6.2(a-b) show two different representations of the same data. Figures depict that there are high differences regarding the total download time depending on the FDT transmission frequency when No-FEC is applied. These differences are higher as losses increase. As Fig. 6.2 reflect, the values of m that provide minimum download times are in the range $[N/10,\ 2N]$ for all percentage of losses. On the other hand, the download time gets worse if a lot of FDT Instances ($m = 50N$) are sent. Furthermore, sending only one FDT Instance per cycle is not the best option either.



(a) Representation 1

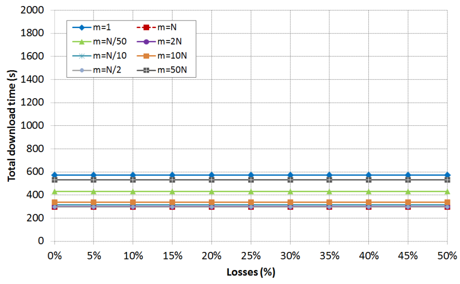


(b) Representation 2

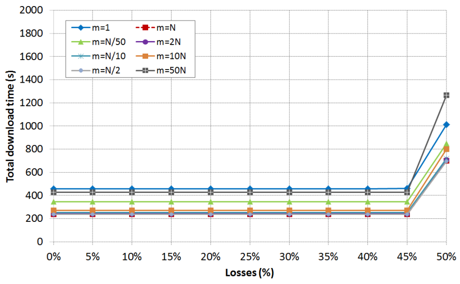
Fig. 6.2. Total download time evaluation for $N = 100$ files and constant file size of $n_i = 1000$ packets, with No-FEC.

The system performance can be improved by using AL-FEC. In this sense, Fig. 6.3(a-f) show the total download time for different values of the code rate (CR). It should be noted that the scale of the total download time is different comparing with Fig. 6.2, as well as the scale of Fig. 6.3(f).

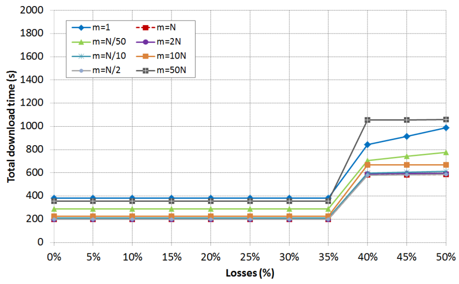
As expected, the use of AL-FEC reduces considerably the total download time. Regarding the FDT frequency, as in Fig. 6.2, values around $m = N$ provide minimum download times for all percentage of losses and code rates, whereas sending only one FDT Instance or a lot of FDT Instances is not a good solution. Fig. 6.3(a-f) reflect in some cases a constant total download time for different percentage of losses, due to the use of enough AL-FEC. When the amount of AL-FEC applied is not enough (high values of code rate) the download time starts increasing (for instance, for losses higher than 10% when the code rate is 0.8) because more carousel cycles are needed to complete the download. Although using a high code rate guarantees a good protection, this does not always entails a minimum value of download time, since using too much parity produces that a lot of parity packets are sent, thus reducing the efficiency of the transmission. Furthermore, a high protection increases considerably the bandwidth. In this example, as the figures show, a code rate that provides a good tradeoff between total download time and bandwidth is 0.7 –Fig. 6.3(d)–, so this code rate will be used in the rest of studies.



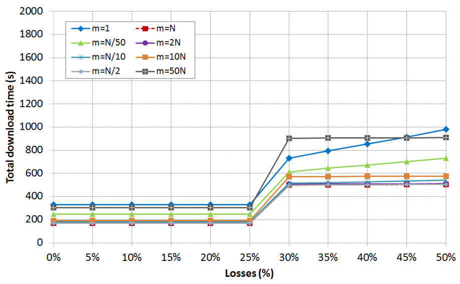
(a) CR = 0.4



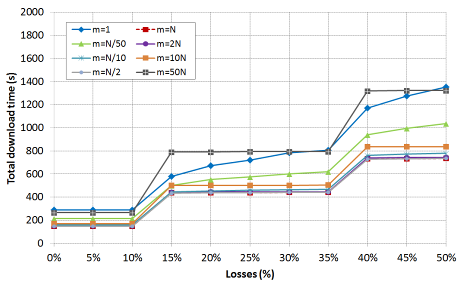
(b) CR = 0.5



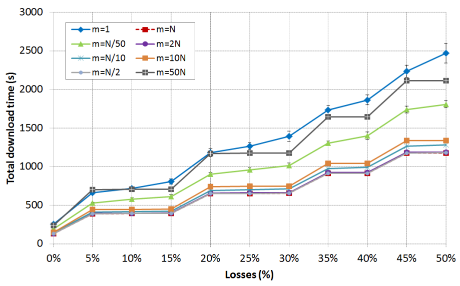
(c) CR = 0.6



(d) CR = 0.7



(e) CR = 0.8



(f) CR = 0.9

Fig. 6.3. Total download time evaluation for N = 100 files and constant file size of n_i = 1000 packets for different values of code rate.

In this sense, Fig. 6.4 depicts the total download time for different percentage of losses and values of *m* for the code rate equal to 0.7. Figure shows that there is a range (between $m = N/10$ and $m = 2N$) where the total download time is minimum.

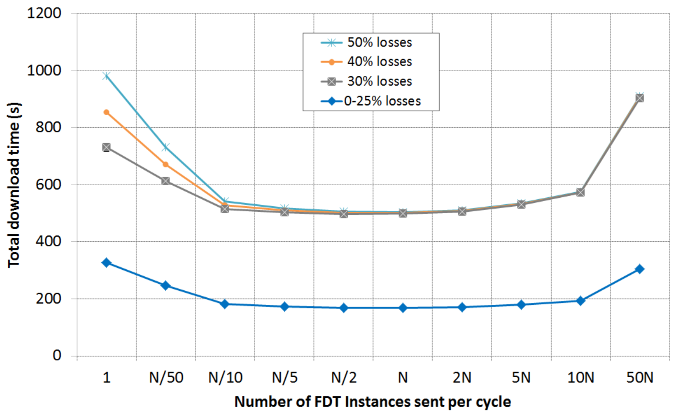
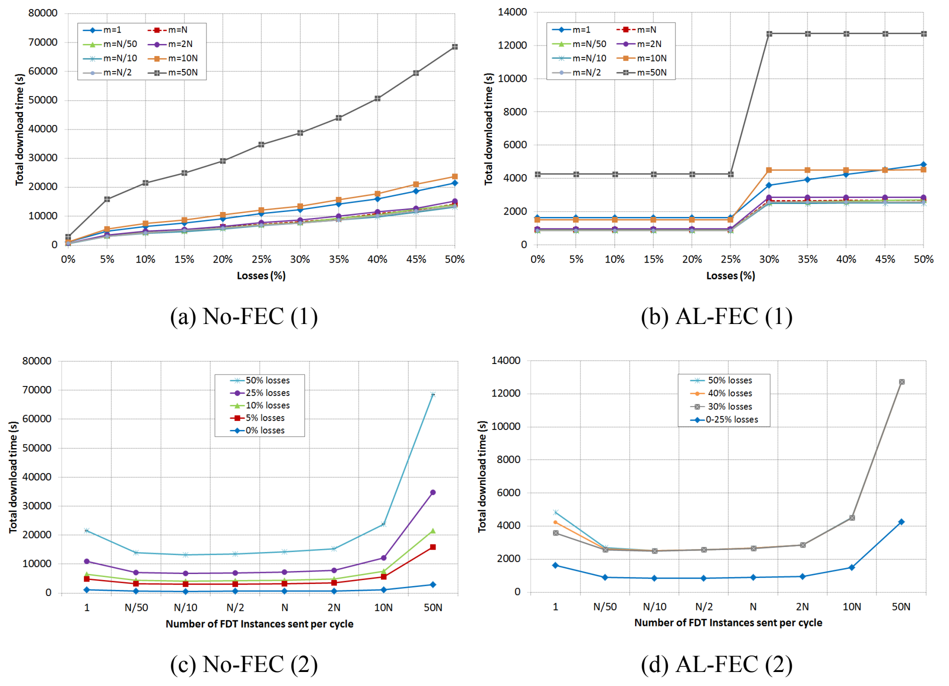


Fig. 6.4. Total download time evaluation for N = 100 files and constant file size of n_i = 1000 packets, with AL-FEC (CR = 0.7).

6.4.1.a. Impact of the number of files in the carousel

Regarding the number of files in the carousel, previous conclusions are confirmed in the studies carried out with low and high values of *N*. The differences between the different values of *m* are higher as the number of files in the carousel increases. To see this, Fig. 6.5(a-d) show the behavior of the FDT when N = 500 files in two different representations. The figures show higher differences between the different values of *m*. Once again, values in range [N/10, 2N] provide minimum download times for all channel losses, whether AL-FEC is applied or not.

Table 6.2 and Table 6.3 show the difference between the total download time obtained with the optimum value ($m = N$) and the rest of values, for different values of *N* and percentage of losses, without using FEC and using AL-FEC respectively. It should be noted that, in the case of $N = 10$, the value of $m = N/50$ has no sense (since $m < 1$), and the value of $m = N/10$ is the same as $m = 1$. Tables show how the optimum value of *m* slightly decreases as the number of files in the carousel increases. Thus, when $N = 10$ or $N = 100$, value of $m = N$ provides the minimum total download time, whereas when $N = 500$, values of *m* lower than *N* provide slightly better results. In any case, there is a range of values of *m* that provide similar and minimal values of the total download time. According to the table, values of *m* in range [N/10, 2N] are optimal for all percentage of losses. Other values increase the total download time considerably. For example, in Table 6.2 and Table 6.3 for $N = 500$ files and $m = 10N$ (that is, sending an FDT Instance each 100 packets) the total download time obtained is over 70% higher than the one obtained with $m = N$ (that is, sending an FDT Instance each 1000 packets). Moreover, as mentioned, sending FDT Instances more frequently increases the total download time considerably, as value of $m = 50N$ shows. Also, sending only one FDT per cycle provides values of total download time rather higher.



(a) No-FEC (1) (b) AL-FEC (1)

(c) No-FEC (2) (d) AL-FEC (2)

Fig. 6.5. Total download time evaluation for N = 500 files and constant file size of n_i = 1000 packets.

Table 6.2. Total download time percentage referred to the case of sending $m = N$ FDT Instances per carousel cycle for files of constant size of n_i = 1000 packets with No-FEC.

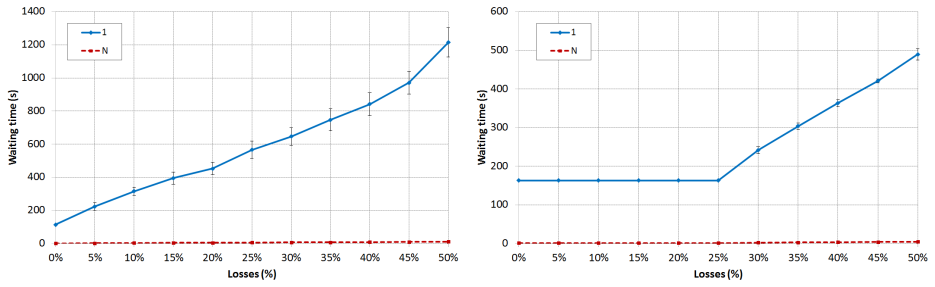
	***N* = 10**			***N* = 100**			***N* = 500**		
	losses			**losses**			**losses**		
m	**5%**	**25%**	**50%**	**5%**	**25%**	**50%**	**5%**	**25%**	**50%**
1	22%	14%	14%	34%	40%	46%	51%	50%	52%
***N*/50**	-	-	-	17%	19%	20%	3%	-2%	-2%
***N*/10**	-	-	-	5%	3%	3%	-2%	-6%	-7%
***N*/5**	10%	7%	7%	1%	-1%	3%	-3%	-6%	-6%
***N*/2**	1%	-1%	1%	1%	-1%	0%	-1%	-5%	-5%
2*N*	1%	-3%	0%	2%	3%	2%	10%	7%	8%
5*N*	2%	-1%	-1%	6%	7%	7%	36%	28%	32%
10*N*	-1%	2%	-1%	16%	15%	16%	78%	68%	68%
50*N*	9%	5%	7%	78%	82%	83%	404%	377%	384%

Table 6.3. Total download time percentage referred to the case of sending $m = N$ FDT Instances per carousel cycle for files of constant size of $n_i = 1000$ packets with AL-FEC.

	$N = 10$			$N = 100$			$N = 500$		
	losses			losses			losses		
m	**5%**	**25%**	**50%**	**5%**	**25%**	**50%**	**5%**	**25%**	**50%**
1	78%	179%	99%	94%	94%	94%	84%	84%	82%
***N*/50**	-	-	-	46%	46%	45%	1%	1%	1%
***N*/10**	-	-	-	7%	7%	7%	-5%	-5%	-5%
***N*/5**	36%	64%	51%	3%	3%	3%	-5%	-5%	-5%
***N*/2**	9%	14%	10%	0%	0%	0%	-4%	-4%	-4%
2*N*	-5%	-9%	-7%	1%	1%	1%	8%	8%	8%
5*N*	-7%	-13%	-9%	6%	6%	6%	31%	31%	31%
10*N*	-7%	-13%	-10%	14%	14%	14%	69%	69%	69%
50*N*	-1%	-8%	-4%	80%	80%	80%	379%	379%	379%

6.4.1.b. Impact of the waiting time

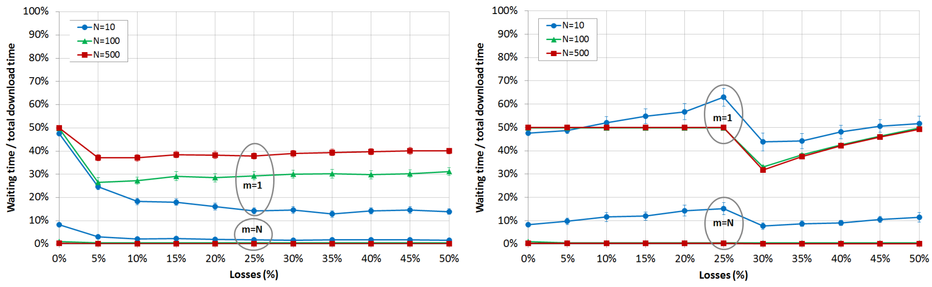
On the other hand, as mentioned in the theoretical analysis, there are two variables that make up the total download time (t_T): the waiting time (t_W) and the download time of the file (t_D). According to the theoretical analysis, the number of FDT Instances sent affects mainly the waiting time. Fig. 6.6(a-b) show the waiting time for only two values of m ($m = 1$ and $m = N$), since values higher than $m = N$ provide values very close to 0. Results are very clear: it is enough to send only N FDT Instances per cycle so as to minimize the waiting time.



(a) No-FEC (b) AL-FEC

Fig. 6.6. Waiting time evaluation for $N = 100$ files and constant file size of $n_i = 1000$ packets.

In this sense, a related study is presented in Fig. 6.7(a-b), where it is shown how the waiting time affects the total download time. Specifically, the figures depict the percentage of t_W regarding t_T for different values of N. Also, the same two values of m of the previous figures are shown. As m is higher, the waiting time hardly affects the total download time. In contrast, when only one FDT is sent in each carousel cycle, the waiting time has a great impact on the total download time.



(a) No-FEC (b) AL-FEC

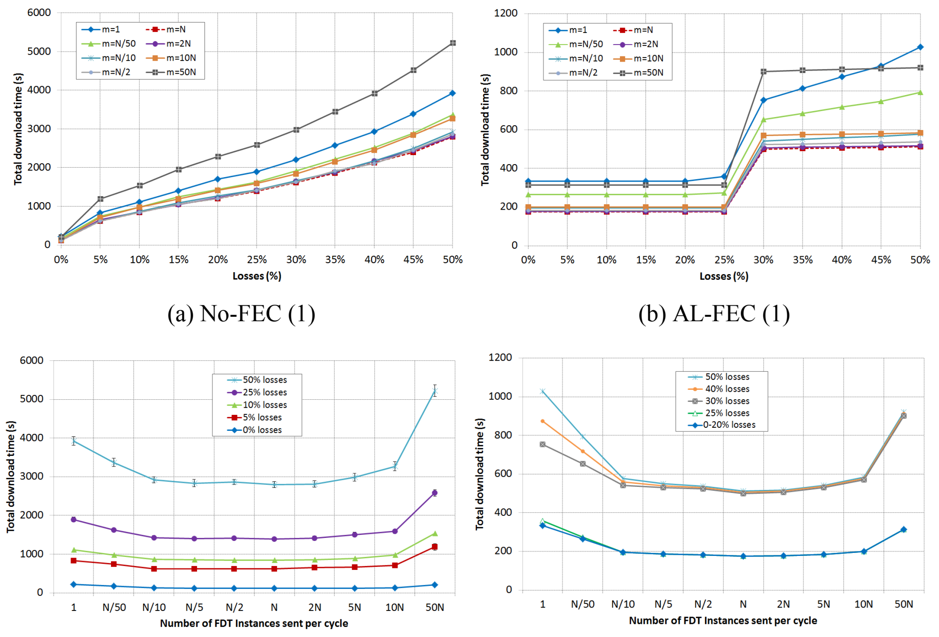
Fig. 6.7. t_W/t_T relation evaluation for $m = 1$ and $m = N$.

When No-FEC is used, the percentage of t_W regarding t_T is higher as the number of files in the carousel increases if only one FDT Instance is sent. For instance, when $N = 500$, t_W represents over 40% of the t_T. Moreover, Fig. 6.7(a) reflects that, for a fixed value of N, the percentage remains very similar independently of the losses, except in the case of no losses. In contrast, if AL-FEC is used the waiting time has more importance on the total download time when the FDT is not sent very frequently. Fig. 6.7(b) shows how the relation t_W/t_T varies depending on the losses. From a certain percentage of losses, the relation t_W/t_T decreases and then increases again. This percentage of losses is delimited by the code rate used. In Fig. 6.7(b), this percentage is over a 25%, a result coherent with Fig. 6.3(d) and Fig. 6.5(b).

For example, if 100 files are sent in a channel with 10% losses and only one FDT per cycle carousel is sent, the waiting time represents 25% of the total download time if No-FEC is used, whereas this percentage increases up to 50% when AL-FEC is used. On the contrary, if N FDTs are sent, the waiting time represents only the 0.5% of the total download time without FEC and the 1% with FEC. This percentage is similar in all cases studied in this section.

6.4.1.c. Impact of the file size distribution

Regarding the file size distribution, normally the files of the carousel will not have the same size. Fig. 6.8 considers a carousel where the size of the files follows a log normal distribution which mean is 1000 packets. Regarding the comparison between different values of m, results are very similar to those shown in Fig. 6.2(a-b), Fig. 6.3(d) and Fig. 6.4. In addition, as Fig. 6.8(a) and Fig. 6.8(c) show, a carousel which file size follows a log normal distribution provides slightly higher download times when No-FEC is used. Nevertheless, when AL-FEC is used –Fig. 6.8(b) and Fig. 6.8(d)– the total download time obtained using a log normal file size distribution is almost the same as the one obtained when files have the same size. Again, it should be noted the different scale of Fig. 6.8(a and c) and Fig. 6.8(b and d).



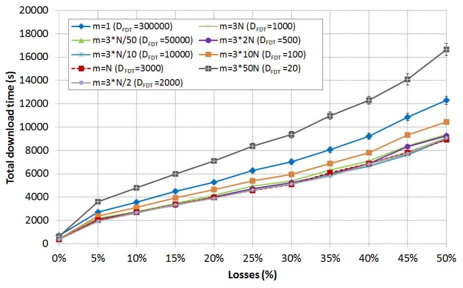
(a) No-FEC (1) (b) AL-FEC (1)

(c) No-FEC (2) (d) AL-FEC (2)

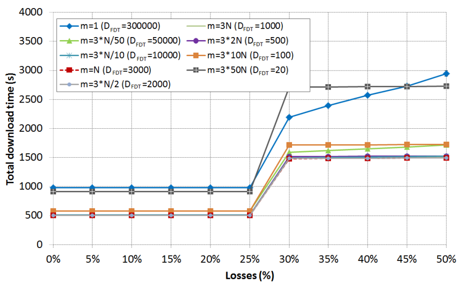
Fig. 6.8. Total download time evaluation for N = 100 files and log normal file size distribution with mean = 1000 packets.

6.4.1.d. Impact of the file size

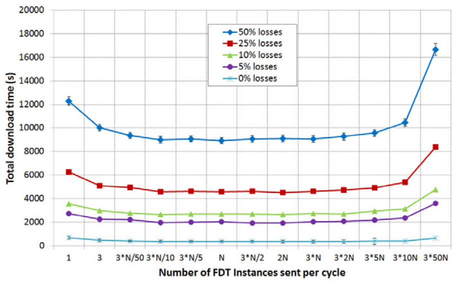
Next study considers a carousel which files are 3 times larger than in previous studies, that is, files of 3000 packets (over 4 MB). In this study, the values of m have been modified in order to maintain the same values of D_{FDT} of the previous studies. Thus, the values of m: 1, 3*N/50, 3*N/10, 3*N/5, N, 3*N/2, 3*N, 3*2N, 3*5N, 3*10N and 3*50N mean sending the FDT Instances every approximately 300 000, 50 000, 10 000, 5000, 3000, 2000, 1000, 500, 200, 100 and 20 packets respectively. Fig. 6.9(a-d) show that, again, there is a wide range of values around $m = N$ that minimizes the total download time for all percentage of losses. These are also the optimum values in a carousel in which the file size of the carousel follows a log normal distribution (although they are not shown here). Therefore, the conclusions are very similar to those reached in the previous studies.



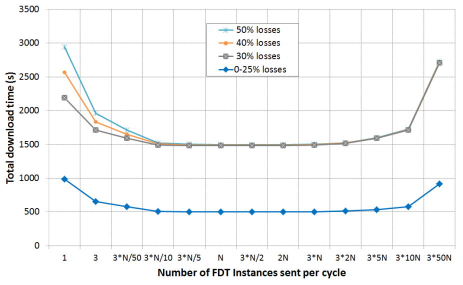
(a) No-FEC (1)



(b) AL-FEC (1)



(c) No-FEC (2)



(d) AL-FEC (2)

Fig. 6.9. Total download time evaluation for N = 100 files and constant file size of n_i = 3000 packets.

6.4.2. Interleaving

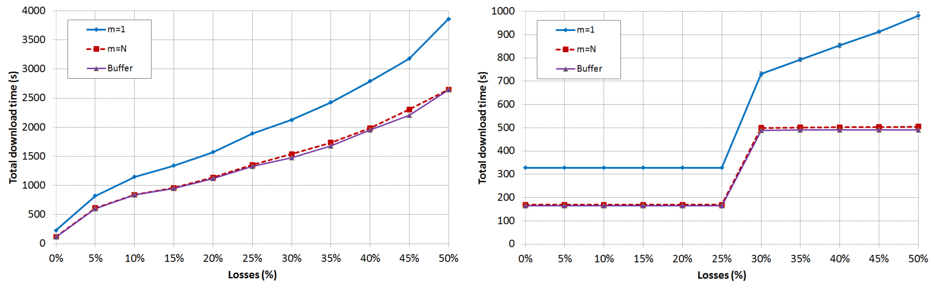
On the other hand, next study analyzes the effect of the scheduling model. Table 6.4 shows the increase (in percentage) of the total download time using interleaving with respect to using sequential scheduling for different percentage of losses, number of FDT Instances sent and using AL-FEC or not. The increase is especially important, apart from the case of no losses without FEC, when AL-FEC is used. In these cases, the difference between sequential and interleaved transmission is approximately half cycle. For instance, in the case of AL-FEC with 25% of losses, it is necessary only one cycle to download the file but, whereas in the sequential mode the download is completed after approximately half cycle, the interleaved transmission requires almost the complete cycle, thus the difference regarding the total download time is almost the double comparing these two models. This fact and the rest of results on the table are coherent with the formulas previously presented. Independently of the number of FDT Instances sent during the carousel, the interleaving model always increases the total download time.

Table 6.4. Relation between the total download time obtained with interleaving regarding the total download time obtained with sequential scheduling. $N = 100$ files and constant file size of $n_i = 1000$ packets.

	No-FEC				AL-FEC			
	losses				losses			
m	**0%**	**5%**	**25%**	**50%**	**0%**	**5%**	**25%**	**50%**
1	92%	29%	10%	4%	24%	28%	49%	17%
***N*/50**	66%	15%	8%	4%	32%	38%	65%	22%
***N*/10**	89%	16%	6%	4%	44%	51%	89%	30%
***N*/5**	93%	20%	9%	2%	46%	54%	93%	31%
***N*/2**	96%	20%	10%	5%	47%	55%	96%	32%
N	181%	38%	14%	8%	48%	56%	97%	33%
2*N*	182%	36%	15%	9%	48%	56%	97%	33%
5*N*	183%	34%	18%	7%	48%	56%	98%	33%
10*N*	183%	35%	15%	7%	48%	56%	98%	33%
50*N*	183%	38%	16%	6%	48%	56%	98%	33%

6.4.3. Prefetching

Finally, it is considered the case of storing packets before the FDT is received. It may appear that this could reduce considerably the total download time but, according to Fig. 6.6(a-b) and equation (6.27), this assumption is only true when few FDT Instances are sent. In this sense, Fig. 6.10(a-b) show a comparison among the total download obtained when a buffer is used and when is not, for two different values of m. It should be noted that, for the sake of clarity, only it is shown one value of m (specifically $m = 1$) when a buffer is used, since other values provide practically the same total download time. Fig. 6.10(a-b) show that it is recommended to use a buffer only when few FDT Instances are sent. For example, if only one FDT is sent, saving packets before the FDT is received can reduce the total download time almost a 50% for different percentages of losses. Nevertheless, the storage space required by clients could increase by a factor equal to the number of files in the carousel, in this case by 100.



(a) No-FEC (b) AL-FEC

Fig. 6.10. Total download time evaluation using a buffer for N = 100 files and constant file size of n_i = 1000 packets.

6.5. Conclusions

This chapter has shown the great influence that the FDT transmission frequency has over the total download time of a file. A proper configuration of the FDT transmission frequency increases the bandwidth efficiency, thus improving the Quality of Experience perceived by the users. In this sense, there are certain configurations of the FDT that reduce the total download time. The results show that there is a range of values of the FDT delivery frequency that minimizes the total download time. Sending as many FDT Instances as files (N) in the carousel per cycle provide very good results, whether AL-FEC is used or not. However, if only one FDT Instance per cycle is sent, the time clients wait until they receive the FDT increases considerably, therefore the total download time increases too. Moreover, if several FDT Instances are sent the total download time also gets worse considerably. The difference among the total download time obtained using an optimum FDT configuration and other configurations is higher as the number of files in the carousel and its size is higher, and as the amount of losses in the channel increases. Thus, in general, there is a range of values around N which provides miminum download times.

Obviously the use of AL-FEC mechanisms improves considerably the total download time. In this sense, it is important to consider the percentage of losses of the transmission channel, in order to use an appropriate code rate that provides a good value of the total download time without increasing excessively the channel bandwidth.

Among the two parameters that make up the total download time, the waiting time has an important impact on the total download time only when few FDT Instances per cycle are sent. Sending more FDT Instances allows to minimize the waiting time. In this sense, it is not a great advantage to save packets before the FDT is received. In doing this, the waiting time would disappear, but as this waiting time would be very low with a proper value of FDT frequency, the profit would be minimal, at the expense of increasing extensively the memory and computational resources in reception.

On the other hand, although interleaved scheduling can be very useful to minimize the burst losses, sequential transmission always reduces the total download time.

To sum up, as a particular conclusion of this chapter and as a general conclusion of this thesis work, a proper configuration of the AL-FEC protection and the FDT provides an optimum content delivery through the FLUTE protocol in multicast wireless networks.

The following publication in an international journal summarizes the results presented in this chapter: [J.8].

References

[1] World Bank, “Information and communications for development 2012: maximizing mobile,” available at: http://www.worldbank.org, accessed: Nov. 2013.

[2] T. Paila, R. Walsh, M. Luby, V. Roca, and R. Lehtonen, “FLUTE – File delivery over unidirectional transport,” *IETF RFC*, vol. 6726, Nov. 2012.

[3] H. T. Chiao, S. Y. Chang, K. M. Li, Y. T. Kuo, and M. C. Tseng, “WiFi multicast streaming using AL-FEC inside the trains of high-speed rails,” presented at the IEEE Int. Symp. on Broadband Multimedia System and Broadcasting (BMSB), Seoul, Korea, Jun. 2012.

[4] J. Peltotalo, J. Harju, and M. Hannuksela, “Reliable, server-friendly and bandwidth-efficient file delivery system using FLUTE server file format,” presented at the IEEE Int. Symp. on Broadband Multimedia System and Broadcasting (BMSB), Bilbao, Spain, May 2009.

[5] E. Paolini, M. Varrella, M. Chiani, B. Matuz, and G. Liva, “Low-complexity LDPC codes with near-optimum performance over the BEC,” in *Proc. Advanced Satellite Multimedia Systems (ASMS)*, Bologna, Italy, Aug. 2008, pp. 274-282.

[6] V. Roca, C. Neumann, and D. Furodet, “Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes,” *IETF RFC*, vol. 5170, Jun. 2008.

[7] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh, “FLUTE – File delivery over unidirectional transport,” *IETF RFC*, vol. 3926, Oct. 2004.

[8] M. Handley, V. Jacobson, and C. Perkins, “SDP – Session description protocol,” *IETF RFC*, vol. 4566, Jul. 2006.

[9] *Digital Video Broadcasting (DVB); IP datacast over DVB-H: content delivery protocols*, ETSI TS 102 472 v1.3.1, Jun. 2009.

[10] M. Luby, M. Watson, and L. Vicisano, “Asynchronous layered coding (ALC) protocol instantiation,” *IETF RFC*, vol. 5775, Apr. 2010.

[11] M. Luby, M. Watson, and L. Vicisano, “Layered coding transport (LCT) building block,” *IETF RFC*, vol. 5651, Oct. 2009.

[12] M. Watson, M. Luby, and L. Vicisano, “Forward error correction (FEC) building block,” *IETF RFC*, vol. 5052, Aug. 2007.

[13] R. Walsh, J. Peltotalo, S. Peltotalo, I. Curcio, and H. Mehta, "SDP descriptors for FLUTE," IETF draft v3, Sept. 2012.

[14] V. Roca, Z. Khallouf, and J. Laboure, "Design and evaluation of a low density generator matrix (LDGM) large block FEC codec," presented at the 5th Int. Workshop on Networked Group Communication (NGC), Munich, Germany, Sep. 2003.

[15] P. Deutsch and J. L. Gailly, "ZLIB compressed data format specification version 3.3," *IETF RFC*, vol. 1950, May 1996.

[16] P. Deutsch, "DEFLATE compressed data format specification version 1.3," *IETF RFC*, vol. 1951, May 1996.

[17] P. Deutsch, "GZIP file format specification version 4.3," *IETF RFC*, vol. 1952, May 1996.

[18] M. Watson, "Basic forward error correction (FEC) schemes," *IETF RFC*, vol. 5445, Mar. 2009.

[19] M. Luby and V. Goyal, "Wave and equation based rate control (WEBRC) building block," *IETF RFC*, vol. 3738, Apr. 2004.

[20] Internet Assigned Number Authority (IANA), "Reliable Multicast Transport (RMT) FEC Encoding IDs and FEC Instances IDs," available at: www.iana.org/assignments/rmt-fec-parameters/rmt-fec-parameters.xhtml, updated Mar. 2012.

[21] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM Journal of Applied Math*, vol. 8, pp. 300-304, 1960.

[22] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo, "Reed-Solomon forward error correction (FEC) schemes," *IETF RFC*, vol. 5510, Apr. 2009.

[23] A. Shokrollahi, "Raptor codes," *IEEE Trans. on Information Theory*, vol. 52, pp. 2551-2567, 2006.

[24] M. Luby, "LT codes," presented at the IEEE Symp. on Foundations of Computer Science (FOCS), Vancouver, Canada, Nov. 2002.

[25] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor forward error correction scheme for object delivery," *IETF RFC*, vol. 5053, Sep. 2007.

[26] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ forward error correction scheme for object delivery," *IETF RFC*, vol. 6330, Aug. 2011.

[27] R. G. Gallager, "Low density parity check codes," *IEEE Trans. on Information Theory*, vol. 8, no. 1, 1962.

[28] D. MacKay and R. Neal, "Good codes based on very sparse matrices," *Lecture notes in Computer Science*, vol. 1025, pp. 100-111, 1995.

[29] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," *Proc. of the ACM Symp. on Theory of Computing (STOC)*, El Paso, Texas, USA, May 1997, pp. 150-159.

[30] S. Park and K. Miller, “Random number generators: good ones are hard to find,” *Communications of the ACM*, vol. 33, no. 1, pp. 87-88, 1990.

[31] V. Zyablov and M. Pinsker, “Decoding complexity of low-density codes for transmission in a channel with erasures,” translated from *Problemy Peredachi Informatsii*, vol. 10, no. 1, pp. 15-28, 1974.

[32] *Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP based Networks (and associated XML)*, ETSI TS 102 034 v1.4.1, Aug. 2009.

[33] *3GPP; Universal Mobile Telecommunications System (UMTS); LTE; Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs (release 11)*, ETSI TS 126 346 v11.5.0, Jul. 2013.

[34] D. Lecompte and F. Gabin, “Evolved multimedia broadcast/multicast service (eMBMS) in LTE-advanced: overview and rel-11 enhancements,” *IEEE Communications Magazine*, vol. 50, no. 11, pp. 68-74, 2012.

[35] C. Neumann, V. Roca, and R. Walsh, “Large scale content distribution protocols,” *ACM Computer Communications Review*, vol. 35, no. 5, pp. 85-92, 2005.

[36] F. Fraile, I. de Fez, and J. C. Guerri, “Modela-TV: service personalization and business model management for mobile TV,” presented at the 7th European Interactive TV Conference (EuroITV), Leuven, Belgium, Jun. 2009.

[37] C. C. Hsieh, C. H. Lin, and W. T. Chang, “Design and implementation of the interactive multimedia broadcasting services in DVB-H,” *IEEE Trans. on Consumer Electronics*, vol. 55, no. 4, pp. 1779-1787, 2009.

[38] J. Peltotalo, J. Harju, M. Saukko, L. Väätämöinen, I. Bouazizi, and I. Curcio, “Personal mobile broadcasting based on the 3GPP MBMS system,” in *Proc. of Int. Conf. on Advances in Mobile Computing and Multimedia (MoMM),* Linz, Austria, Nov. 2008, pp. 156-162.

[39] L. Liang, H. Cruichkshank, Z. Sun, C. Kulatunga, and G. Fairhurst, “The integration of TESLA and FLUTE over satellite networks,” presented at the IEEE Global Telecommunications Conference (GLOBECOM), Miami, FL, USA, Dec. 2010.

[40] D. Kutscher, J. Greifenberg, and K. Loos, “Scalable DTN distribution over unidirectional links,” presented at the SIGCOMM Workshop on Networked Systems in Developing Regions (NSDR), Kyoto, Japan, Aug. 2007.

[41] G. Papastergiou, I. Psaras, and V. Tsaoussidis, “Deep-space transport protocol: a novel transport scheme for space DTNs,” *Computer Communications*, vol. 32, no. 16, pp. 1757-1767, 2009.

[42] J. Hrvoje, T. Stockhammer, W. Xu, and W. Abdel Samad, “Efficient video-on-demand services over mobile datacast channels,” *Journal of Zhejiang University*, vol. 7, no. 5, pp. 873-884, 2006.

[43] Z. Yetgin and T. Çelik, "Efficient progressive downloading over multimedia broadcast multicast service," *Computer Networks*, vol. 56, no. 2, pp. 533-547, 2012.

[44] C. Neumann and V. Roca, "Scalable video streaming over ALC (SVSoA): a solution for the large scale multicast distribution of videos," presented at the 1st Int. Workshop on SMDI, Athens, Greece, May 2004.

[45] J. Peltotalo, S. Peltotalo, J. Harju, and R. Walsh, "Performance analysis of a file delivery system based on the FLUTE protocol," *Int. Journal of Communication Systems*, vol. 20, no. 6, pp. 633-659, 2007.

[46] MAD-FCL, MAD Project's home page, available at: http://mad.cs.tut.fi, accessed: Nov. 2013.

[47] J. Peltotalo, J. Harju, and M. M. Hannuksela, "Reliable, server-friendly and bandwidth-efficient file delivery system using FLUTE server file format," presented at the IEEE Int. Symp. on Broadband Multimedia Systems and Broadcasting (BMSB), Bilbao, Spain, May. 2009.

[48] C. Neumann and V. Roca, "Impacts of the startup behavior of multilayered multicast congestion control protocols on the performance of content delivery protocols," in *Proc. of the Int. Workshop on Web Content Caching Distribution (WCW)*, Sophia Antipolis, France, Sep. 2005, pp. 144-150.

[49] C. Neumann, V. Roca, A. Francillon, and D. Furodet, "Impacts of packet scheduling and packet loss distribution on FEC performances: observations and recommendations," in *Proc. of the ACM Conf. on Emerging network experiment and technology (CoNEXT)*, Toulouse, France, Oct. 2005, pp. 166-176.

[50] INRIA Planète Research Team, LDPC large block FEC codec distribution, available at: http://planete-bcast.inrialpes.fr/article.php3?id_article=16, accessed: Nov. 2013.

[51] V. Roca and C. Neumann, "Design, evaluation and comparison of four large block FEC codecs, LDPC, LDGM, LDGM staircase and LDGM triangle, plus a Reed-Solomon small block FEC codec," INRIA Research Report RR-5225, Jun. 2004.

[52] E. Paolini, M. Varrella, M. Chiani, B. Matuz, and G. Liva, "Low-complexity LDPC codes with near-optimum performance over the BEC," in *Proc. Advanced Satellite Mobile Systems (ASMS)*, Bologna, Italy, Aug. 2008, pp. 274-282.

[53] M. Cunche and V. Roca, "Optimizing the error recovery capabilities of LDPC-staircase codes featuring a Gaussian elimination decoding scheme," presented at the 10th IEEE Int. Workshop on Signal Processing for Space Communications (SPSC), Rhodes Island, Greece, Oct. 2008.

[54] M. Cunche and V. Roca, "Improving the decoding of LDPC codes for the packet erasure channel with a hybrid Zyablov iterative decoding/Gaussian elimination Scheme," INRIA Research Report RR-6473, Mar. 2008.

[55] M. Cunche, V. Savin, and V. Roca, "Analysis of Quasi-Cyclic LDPC codes under ML decoding over the erasure channel," in *Proc. of the IEEE Int. Symp. on*

Information Theory and its Applications (ISITA), Taichung, Taiwan, Oct. 2010, pp. 861-866.

[56] V. Roca, A. Roumy, and B. Sayadi, "The generalized object encoding (GOE) LDPC-staircase FEC scheme," *IETF draft*, Jul. 2012.

[57] H. Bai and M. Atiquzzaman, "Error modeling schemes for fading channels in wireless communications: A survey," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 2, pp. 2-9, 2003.

[58] Cisco Systems webpage, "Industry solutions: Cisco StadiumVision Mobile," available at: http://www.cisco.com/web/strategy/sports/stadiumvision_mobile.html, accessed Nov. 2013.

[59] R. Belda, I. de Fez, F. Fraile, V. Murcia, P. Arce, and J. C. Guerri, "Multimedia System for Emergency Services over TETRA-DVBT Networks," presented at the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Parma, Italy, Sep. 2008.

[60] K. Chorianopoulos, "Personalized and mobile digital TV applications," *Multimedia tools and applications*, vol. 36, no. 1-2, pp. 1-10, 2008.

[61] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4-7, 2001.

[62] J. W. Streefker, M. P. van Esch-Bussemakers, and M. A. Neerincx, "Designing personal attentive user interfaces in the mobile public safety domain," *Computers in Human Behavior*, no. 22, pp. 749-770, 2006.

[63] D. S. Weld, C. Anderson, P. Domingos, O. Etzioni, K. Gajos, T. Lau, and S. Wolfman, "Automatically personalizing user interfaces," in *Proc. Int. Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, Aug. 2003, pp. 1613-1619.

[64] C. Chatfield, D. Carmichael, R. Hexel, J. Kay, and B. Kummerfeld, "Personalisation in intelligent environments: managing the information flow," in *Proc. of the CHISIG Conference on Human Computer Interaction (OZCHI)*, Canberra, Australia, Nov. 2005, pp. 1-10.

[65] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263-277, 2007.

[66] M. Valtonen, A. M. Vainio, and J. Vanhala, "Proactive and adaptive fuzzy profile control for mobile phones," presented at the IEEE Int. Conf. on Pervasive Computing and Communications (PerCom), Galveston, Texas, USA, Mar. 2009.

[67] P. Korpipaa, E. J. Malm, T. Rantakokko, V. Kyllonen, J. Kela, J. Mantyjarvi, J. Hakkila, and I. Kansala, "Customizing user interaction in smart phones," *IEEE Pervasive Computing*, vol. 5, no. 3, pp. 82-90, 2006.

[68] A. Gil, F. Fraile, M. Ramos, I. de Fez, and J. C. Guerri, "Personalized Multimedia Touristic Services for Hybrid broadcast/broadband Mobile receivers," *IEEE Trans. on Consumer Electronics*, vol. 56, no. 1, pp. 211-219, 2010.

[69] F. Fraile, I. de Fez, and J. C. Guerri, “Evaluation of background push content download services to mobile devices over DVB networks,” *IEEE Trans. on Broadcasting*, doi: 10.1109/TBC.2013.2289639, 2013.

[70] T. Stockhammer and M. G. Luby, “DASH in mobile networks and services,” presented at the IEEE Visual Communications and Image Processing (VCIP), San Diego, CA, USA, Nov. 2012.

[71] ISO/IEC 23009-1, “Dynamic adaptive streaming over HTTP (DASH) – Part 1: media presentation description and segment formats,” 2012.

[72] P. Seeling and M. Reisslen, “Video transport evaluation with H.264 video traces,” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1142-1165, 2012.

[73] C. H. Lin, Y. C. Wang, C. K. Shieh, W. S. Hwang, “An unequal error protection mechanism for video streaming over IEEE 802.11e WLANs,” *Computer Networks*, vol. 56, no. 11, pp. 2590-2599, 2012.

[74] A. Downey, “The structural cause of file size distributions,” in *Proc. of the 9th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Cincinnati, OH, USA, Aug. 2001, pp. 361-370.

[75] *Digital Video Broadcasting (DVB); Transport of MPEG-2 TS based DVB Services over IP based Networks*, ETSI TS 102 034 v1.4.1, Aug. 2009.

[76] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, “A digital fountain approach to reliable distribution of bulk data,” in *Proc. of the ACM SIGCOMM Conf. on Applications, Technologies, Architectures and Protocols for Computer Communication*, Vancouver, Canada, Sep. 1998, pp. 56-67.

[77] I. de Fez, F. Fraile, R. Belda, and J. C. Guerri, “Performance evaluation of AL-FEC LDPC codes for push content applications in wireless unidirectional environments,” *Multimedia Tools and Applications*, Springer Netherlands, vol. 60, no. 3, pp. 669-688, 2012.

[78] I. Busse, B. Deffner, and H. Schulzrinne, “Dynamic QoS control of multimedia applications based on RTP,” *Computer Communications*, vol. 19, no. 1, pp. 49-58, 1996.

[79] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey, “Extended RTP profile for real-time transport control protocol (RTCP) –based feedback (RTP/AVPF),” *IETF RFC*, vol. 4585, Jul. 2006.

[80] I. de Fez, F. Fraile, R. Belda, and J. C. Guerri, “Analysis and evaluation of adaptive LDPC AL-FEC codes for content download services,” *IEEE Trans. on Multimedia*, vol. 14, no. 3, pp. 641-650, 2012.

[81] J. C. Guerri, M. Esteve, C. Palau, and V. Casares, “Feedback flow control with hysteresial techniques for multimedia retrievals,” *Multimedia Tools and Applications*, vol. 13, no. 3, pp. 307-332, 2001.

[82] M. Zúñiga-Zamalloa and B. Krishnamachari, “An analysis of unreliability and asymmetry in low-power wireless links,” *ACM Trans. on Sensor Networks (TOSN)*, vol. 3, no. 2, art. 7, 2007.

[83] IEEE Computer Society, “Std 802.11b, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band,” Sep. 1999.

[84] T. S. Rappaport, “Wireless communications. Principles & Practice,” Prentice Hall Communications Engineering and Emerging Technologies Series, 1996.

[85] B. Liang and Z. Haas, “Predictive distance-based mobility management for PCS networks,” in *Proc. of Int. Conf. on Computer Communications (INFOCOM)*, vol. 3, New York, NY, USA, Mar. 1999, pp. 1377-1384.

[86] S. Akshsabi, S. Narayanaswamy, A. C. Begen, and C. Dovrolis, “An experimental evaluation of rate-adaptive video players over HTTP,” *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 271–287, 2012.

Appendix A. List of Publications

Following there is a list of the publications that provide most of the content of this thesis dissertation. The following notation is used: B refers to book chapters, C to conference papers and J to journal papers.

Chapter 2

[J.1] A. Gil, F. Fraile, M. Ramos, I. de Fez, and J. C. Guerri, "Personalized multimedia touristic services for hybrid broadcast/broadband mobile receivers," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 1, pp. 211-219, 2010.

[J.2] F. Fraile, I. de Fez, and J. C. Guerri, "Evaluation of background push content download services to mobile devices over DVB networks," *IEEE Transactions on Broadcasting*, vol. 60, no. 1, pp. 1-15, 2014.

[J.3] R. Belda, I. de Fez, F. Fraile, P. Arce, and J. C. Guerri, "Hybrid FLUTE/DASH video delivery over mobile wireless networks," accepted for publication in *Transactions on Emerging Telecommunications Technologies*, doi: 10.1002/ett.2804, 2014.

[J.4] I. de Fez, M. Gil, J. Fons, J. C. Guerri, and V. Pelechano, "A personalized system for scalable distribution of multimedia content in wireless networks," accepted with revisions in *Multimedia Tools and Applications*, 2014.

[C.1] R. Belda, I. de Fez, F. Fraile, V. Murcia, P. Arce, and J. C. Guerri, "Multimedia System for Emergency Services over TETRA-DVBT Networks," presented at the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Parma, Italy, Sep. 2008.

[C.2] F. Fraile, I. de Fez, and J. C. Guerri, "Modela-TV: Service personalization and business model management for mobile TV," presented at the 7th European Interactive TV Conference (EuroITV), Leuven, Belgium, Jun. 2009.

[C.3] A. Gil, F. Fraile, M. Ramos, I. de Fez, and J. C. Guerri, "Personalized multimedia touristic services for hybrid broadcast/broadband mobile receivers," in *Proc. of the International Conference on Consumer Electronics*, Las Vegas, NV, USA, Jan. 2010, pp. 157-158.

[C.4] F. Fraile, I. de Fez, R. Belda, and J. C. Guerri, “Evaluation of a background push download service for personal multimedia devices,” in *Proc. of the International Conference on Consumer Electronics*, Las Vegas, NV, USA, Jan. 2011, pp. 231-232.

[C.5] T. R. Vargas, P. Arce, I. de Fez, V. Murcia, F. Fraile, R. Belda, P. Acelas, and J. C. Guerri, “Solutions to improve the video streaming service over heterogeneous networks,” presented at the 1st Workshop on Future Internet: Efficiency in High-Speed Networks (W-FIERRO), Cartagena, Spain, Jul. 2011.

Chapter 3

[J.5] I. de Fez, F. Fraile, R. Belda, and J. C. Guerri, “Performance evaluation of AL-FEC LDPC codes for push content applications in wireless unidirectional environments,” *Multimedia Tools and Applications*, Springer Netherlands, vol. 60, no. 3, pp. 669-688, 2012.

[C.6] I. de Fez, F. Fraile, R. Belda, and J. C. Guerri, “Implementación y evaluación de la codificación LDPC para la transmisión de ficheros en entornos unidireccionales,” presented at the Jornadas de Ingeniería Telemática (JITEL), Valladolid, Spain, Sep. 2010.

Chapter 4

[J.6] I. de Fez, F. Fraile, R. Belda, and J. C. Guerri, “Analysis and evaluation of adaptive LDPC AL-FEC codes for content download services,” *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 641-650, 2012.

[C.7] I. de Fez, F. Fraile, R. Belda, and J. C. Guerri, “Evaluation of adaptive LDPC AL-FEC codes for content download services,” presented at the IEEE International Conference on Multimedia and Expo (ICME), Barcelona, Spain, Jul. 2011.

Chapter 5

[J.7] I. de Fez and J. C. Guerri, “An adaptive mechanism for optimal content download in wireless networks,” accepted for publication in *IEEE Transactions on Multimedia*, doi: 10.1109/TMM.2014.2307155, 2014.

Chapter 6

[J.8] I. de Fez, F. Fraile, and J. C. Guerri, “Effect of the FDT transmission frequency for an optimum content delivery using the FLUTE protocol,” *Computer Communications*, vol. 36, no. 12, pp. 1298-1309, 2013.

Additionally, the author has participated in the following publications:

[B.1] F. Fraile, P. Arce, R. Belda, I. de Fez, and J. C. Guerri, “Social aware TV content delivery over intelligent networks,” *Social Media Retrieval*, Ed. Springer, pp. 373-391, 2013.

[J.9] P. Acelas, P. Arce, W. Castellanos, R. Belda, I. de Fez, F. Fraile, V. Murcia, T. R. Vargas, M. Monfort, and J. C. Guerri, “An interactive wireless-based telemedicine system for back care applications,” *Waves 2010 (iTEAM UPV Journal)*, vol. 2, pp. 56-65, 2010.

[J.10] R. Belda, P. Arce, I. de Fez, F. Fraile, and J. C. Guerri, “Android real-time audio communications over local wireless,” *Waves 2012 (iTEAM UPV Journal)*, vol. 4, pp. 35-42, 2012.

[J.11] A. Miquel, R. Belda, I. de Fez, P. Arce, F. Fraile, J. C. Guerri, F. Martínez, and S. Gallardo, “A power consumption monitoring, displaying and evaluation system for home devices,” *Waves 2013 (iTEAM UPV Journal)*, vol. 5, pp. 5-13, 2013.

[C.8] R. Belda, J. M. Oztarman, I. de Fez, and J. C. Guerri, “Señalización SMS para el acceso a redes sociales,” presented at the Telecom I+D 2010, Valladolid, Spain, Sep. 2010.

[C.9] I. de Fez, J. Arambarri, P. Arce, F. Arribas, S. Barrera, I. Bilbao, E. Burgoa, J. C. Guerri, P. Ortiz, E. Vaz, and D. Zaragoza, “GrafiTV: interactive and personalized information system over audiovisual content,” in *Proc. of the NEM Summit*, Istanbul, Turkey, Oct. 2012, pp. 41-46.

Appendix B. List of Projects

During the elaboration of their thesis, the author has participated in the following projects:

[P.1] Modela-TV: Personalización de servicios y gestor de modelos de negocio en TV móvil. FIT-330300-2007-15.

[P.2] MIQUEL: Sistema multimedia sobre entornos inalámbricos aplicado a los trastornos del sistema músculo-esquelético. TEC-2007-68119-C02-01/TCM.

[P.3] Redes híbridas para la provisión de servicios turísticos. TSI-020302-2008-94 / TSI-020302-2010-165.

[P.4] AV-MOV: Contenidos multimedia para el transporte multimodal. TSI-020301-2009-11.

[P.5] GrafiTV: Sistema de información interactiva y personalizada sobre contenidos audiovisuales. TSI-090100-2011-173.

[P.6] HAUS: Hogar digital y contenidos audiovisuales adaptados a los usuarios. IPT-2011-1049-4300000-AR.

[P.7] COMINN: Centro comercial interactivo con interacción natural. IPT-2012-0883-430000.

[P.8] Aplicación para comunicaciones por voz a través de bluetooth y Wi-Fi en dispositivos con sistema operativo Android. Proyectos de prueba de concepto, INNOVA 2012 (Universitat Politècnica de València).

[P.9] Distribución escalable de contenidos multimedia adaptados a las preferencias y contexto de los usuarios. Proyectos de investigación multidisciplinares 2012 (Universitat Politècnica de València). PAID-05-12.

[P.10] ICARE: Innovative Cloud Architecture for Real Entertainment (2012-2015). 11012, ITEA 2 Call 6 Program of the European Union.

...and so is this.