N^{o} assigned by the library

|_|_|_|_|_|_|_|_|_|_|

THESIS

for obtaining the degree of

Doctor of the Université Joseph Fourier

Speciality: ≪ Computer Science ≫

prepared at INRIA Rhône Alpes, Planète Research Team

presented and publicly discussed

by

Christoph NEUMANN

Defended on December 14th, 2005

Title:

Large Scale Content Distribution applied to Files and Videos

Supervisor: Vincent ROCA

Committee in charge

Prof.	Sacha KRAKOWIAK	President
Prof.	Ernst BIERSACK	Reporter
Prof.	Patrick SÉNAC	Reporter
Prof.	Andrzej DUDA	Director of thesis
Dr.	Vincent ROCA	Supervisor
Prof.	Jean-Jacques PANSIOT	Examiner

N° attribué par la bibliothèque

|__|__|__|__|__|__|__|__|__|__|

THÈSE

pour obtenir le grade de

Docteur de l'Université Joseph Fourier

Spécialité: ≪Informatique≫

préparée à l'INRIA Rhône Alpes, projèt Planète

présentée et soutenue publiquement

par

Christoph NEUMANN

le 14.Décembre 2005

Titre:

Diffusion de Contenu à Grande Échelle appliquée aux Fichiers et aux Vidéos

Encadrament scientifique: Vincent ROCA

Jury

Prof.	Sacha KRAKOWIAK	Président
Prof.	Ernst BIERSACK	Rapporteur
Prof.	Patrick SÉNAC	Rapporteur
Prof.	Andrzej DUDA	Directeur de thèse
Dr.	Vincent ROCA	Encadrament scientifique
Prof.	Jean-Jacques PANSIOT	Examinateur

Abstract

Reliable multicast is certainly one of the most effective solutions to distribute content, like files and videos, to a very large number (i.e. possibly millions) of clients. To that respect the ALC and FLUTE protocols, both coming from the IETF Reliable Multicast Transport (RMT) working group, have recently been adopted in the context of 3G cellular networks in 3GPP/MBMS and for DVB-H IP Datacast services.
This work focuses on reliable multicast with massive scalability as a core requirement, and it builds upon the RMT IETF solutions. These reliable multicast protocols rely on several building blocks that we investigate in detail:

- *Forward Error Correction (FEC) Building Block:* We examine the broad class of "Low Density Parity Check" (LDPC) large block FEC codes. We design derivates, namely LDGM-Staircase and LDGM-Triangle, and benchmark these codes in detail with respect to their error correcting capacities, memory requirements and decoding/encoding speeds, and compare them to the Reed Solomon small block code. We find that LDPC codes and their implementation have very promising performance, especially when used with large files.

  The FEC building block must be adapted to the channel, and we experimentally evaluate how to best schedule packets, and derive some recommendations.

  Finally, we analyze the codes with respect to their ability to offer a partial reliability service, i.e. their ability to decode parts of the content even if the whole decoding process cannot finish because too many packet are missing. This study leads to quite surprising results.

- *Congestion Control Building Block:* We look at the startup behavior of the three congestion control protocols RLC, FLID-SL and WEBRC. We show that the startup phase of these protocols significantly impacts the performance of a file download application.

This thesis has also several contributions at application level:

- *FLUTE extensions:* The larger the content, the more efficient FEC protection is. We therefore propose a mechanism to aggregate several files in the file delivery protocol FLUTE, and also explain how to logically group files.

- *Video Streaming:* Designing a video streaming solution for an environment having no or a constrained back channel, due for instance to scalability requirements, leads to many challenges. We introduce the SVSoA version 1 and version 2 approaches, based on ALC. The first one requires hierarchically encoded videos, while the latter is more adapted to a wireless broadcast use case and does not need hierarchical encoded videos anymore. These approaches benefits from all the ALC reliable multicast protocols assets in terms of unlimited scalability, congestion control and error recovery.

Keywords: Reliable Multicast, FLUTE, ALC, Forward Error Correction (FEC), Low Density Parity Check (LDPC) Codes, Content Delivery

Résumé

Le multicast fiable est certainement la solution la plus efficace pour la distribution de contenu à un très grand nombre (potentiellement des millions) de récepteurs. Dans cette perspective les protocoles ALC et FLUTE, standardisés à l'IETF (RMT WG), ont été adoptés dans 3GPP/MBMS et dans le DVB-H IP-Datacast dans les contextes des réseaux cellulaires 3G.
Ce travail se concentre sur le multicast fiable et a comme requis principal le passage l'échelle massif en terme de nombre de clients. Cette thèse se base sur les solutions proposées à l'IETF RMT WG. Ces protocoles de multicast fiable sont construit autour de plusieurs briques de base que nous avons étudié en détail:

- *La brique Forward Error Correction (FEC) :*

  Nous examinons la classe de codes grands blocs Low Density Parity Check (LDPC). Nous concevons des dérivées de ces codes, et les analysons en détail. Nous en concluons que les codes LDPC et leur implémentation ont des performances très prometteuses, surtout si ils sont utilisées avec des fichiers de taille importante.

- *La brique contrôle de congestion :*

  Nous examinons le comportement dans la phase de démarrage de trois protocoles de contrôle de congestion RLC, FLID-SL, WEBRC. Nous démontrons que la phase de démarrage a un grand impact sur les performances de téléchargement.

Cette thèse a aussi plusieurs contributions au niveau applicatif:

- *Extensions de FLUTE :*

  Nous proposons un mécanisme permettant d'agréger plusieurs fichiers dans le protocole FLUTE. Ceci améliore les performance de transmission.

- *Streaming vidéo :*

  Nous proposons SVSoA, une solution de streaming basé sur ALC. Cette approche bénéficie de tout les avantages de ALC en terme de passage à l'échelle, contrôle de congestion et corrections d'erreurs.

Mots clés : Multicast fiable, FLUTE, ALC, codes correcteur d'erreurs, Forward Error Correction (FEC), Low Density Parity Check (LDPC) Codes, diffusion de contenu

Our Achievements: Published Papers, Internet-Drafts, Implementations and Tutorials

Journal Papers

1. Christoph Neumann, Vincent Roca, Rod Walsh, Large Scale Content Distribution Protocols, ACM Computer Communication Review (CCR), Vol. 35 No. 5, Invited Paper, October 2005.

Conference Papers

1. Christoph Neumann, Vincent Roca, Aurélien Francillon, David Furodet, Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances: Observations and Recommendations, ACM CoNEXT'05 Conference, Toulouse, France, October 2005.
   An extended version is available as INRIA Research Report number RR-5578, INRIA Rhone-Alpes, France, May 2005.
2. Christoph Neumann and Vincent Roca, Impacts of the Startup Behavior of Multilayered Multicast Congestion Control Protocols on the Performance of Content Delivery Protocols, IEEE 10th International Workshop on Web Content Caching and Distribution (WCW 2005), Sophia Antipolis, French Riviera, France, September 2005.
3. Christoph Neumann and Vincent Roca, Analysis of FEC Codes for Partially Reliable Media Broadcasting Schemes, 2nd International Workshop on Multimedia Interactive Protocols and Systems (MIPS'04), Best Student Paper Award, Grenoble, France, November 2004.
4. Claude Castelluccia, Gabriel Montenegro, Julien Laganier, and Christoph Neumann, "Hindering Eavesdropping via IPv6 Opportunistic Encryption", In Proc. of 9th European Symposium on Research in Computer Security (ESORICS 2004). Lecture Notes in Computer Science (LNCS), Springer-Verlag, September 2004.
5. Christoph Neumann and Vincent Roca, Scalable Video Streaming over ALC (SVSoA): a Solution for the large Scale Multicast Distribution of Videos, 1st International Workshop on 'Streaming media distribution over the Internet' (SMDI04), Athens, Greece, May 2004.
   An extended version is available as INRIA Research Report number RR-4769, INRIA Rhone-Alpes, France, March 2003.
6. Christoph Neumann and Vincent Roca, Multicast Streaming of Hierarchical MPEG4 Presentations, ACM Multimedia 2002 (Short Paper), Juan Les Pins, France, December 2002.

Reseach Reports

1. Vincent Roca and Christoph 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 number RR-5225, June 2004.

Standardization and Internet Drafts

1. Vincent Roca, Christoph Neumann, David Furodet, Low Density Parity Check (LDPC) Forward Error Correction, IETF RMT Working Group, draft-roca-rmt-fec-bb-ldpc-00.txt (Work in Progress), June 2005.
2. Christoph Neumann, Vincent Roca, Rod Walsh, File Aggregation Scheme for FLUTE, IETF RMT Working Group, draft-neumann-rmt-flute-file-aggregation-01.txt (Work in Progress), June 2005.
   Previous versions (obsoleted): version 00, December 2004
3. Pascal Moniot, Vincent Roca, Christoph Neumann, David Furodet, LDPC Staircase FEC codes, DVB-H proposal, September 2005.

Tutorials

1. Christoph Neumann and Vincent Roca, A Survey of Large Scale Multimedia Streaming Techniques for the Internet, International Workshop on Multimedia Interactive Protocols and Systems (MIPS'03), Napoli, Italy, November 18-21, 2003. http://mips2003.idms-proms.org/

Implementations

1. Vincent Roca and al., MCLv3: an Open Source GNU/GPL Implementation of the ALC/FLUTE and NORM Reliable Multicast Protocols,
   Involvement: 40% of FLUTE/ALC code,
   URL: http://www.inrialpes.fr/planete/people/roca/mcl/.
2. Christoph Neumann, Vincent Roca and Julien Laboure, An Open-Source Implementation of a LDPC/LDGM Large Block FEC Codec,
   Involvement: 33% of LDPC code,
   URL: http://www.inrialpes.fr/planete/people/roca/mcl/.
3. Christoph Neumann, Design of an Streaming Application over ALC (SVSoA),
   Involvement: 100% of code.
4. Christoph Neumann, Julien Laganier, Claude Castelluccia and Gabriel Montenegro, An Open-Source Implementation of Opportunistic Encryption,
   Involvement: 50% of Opportunistic Encryption code,
   URL: http://www.inrialpes.fr/planete/people/chneuman/OE.html.
5. Contributions to an RTP/UDP/IPv4/6 RoHC (Robust Header Compression) profile.

Awards

1. MIPS'04 Best Student Paper Award, International Workshop on Multimedia Interactive Protocols and Systems, Grenoble, France, November 2004.

Acknowledgements

While I was enjoying my time preparing my Ph.D., many people encouraged me, with their support and suggestions.

To all of those, thank you.

Contents

Chapter 1

Introduction

Contents

1.1 Large Scale Content Distribution: Introduction

Large scale content distribution, where the same content is sent reliably to a large number of users, is a challenging and complex task. A point-to-point approach, i.e. an approach where the content is sent over one dedicated unicast connection for each receiver, increases signaling and media-load overhead linearly with group size and is not efficient and generally infeasible for this purpose. The only proven scalable solution is reliable multicast. This subject has been intensively studied, in particular at IETF (section 1.2).

However, for various reasons, listed in [23], multicast routing deployment is far behind expectations (e.g. no ubiquitous multicast routing is available and commercial IPv6 multicast router support is minimal). Wide scale deployment of reliable multicast technologies in the public Internet has not yet taken off.

There are a few exceptions though, in particular in national/international research networks like the MBONE [24] and M6BONE [2] (the reliable multicast file delivery application FLUTE [82] is already used within the M6BONE network), and within sites where multicast routing is easily deployed, and in very specific environments (e.g. in clusters).

Yet until recently a killer-application appealing to the commercial mass market was still missing. It may change with the advent of wireless radio networks that are by nature broadcast networks, and a renewed commercial interest in mobile TV. In particular, new generations of mobile devices (such as cellular phones) will undoubtedly dominate the future Internet [34], while taking advantage of technologies primarily designed for the Internet. More specifically, the IP-Datacast (IPDC) service for DVB-H and the Multimedia Broadcast Multicast Service (MBMS) for the 3G cellular systems are two such enabling systems, and both of them rely on the outcomes of the IETF working groups.

1.1.1 Challenges

Reliable data transfer to a multicast group is difficult since it faces the following challenges:

- *Scalability:* Support for a high number of simultaneous receivers is essential (up to thousands or millions of receivers in some use cases). Ideally the number of receivers should not reduce system performance.
- *Channel and Client Heterogeneity:* The set of receivers and channels will be very heterogeneous, i.e. with very different bandwidths, processing capacities and loss patterns.
- *Content Heterogeneity:* Mass media content is diverse and variable - especially over time to fashion and commercial trends and so the transport solution needs to support any kind of content. Nevertheless this paper only focuses on files and other static contents.
- *Reliability:* The content delivery approach must be robust to each kind of packet losses and cope with intermittent connectivity, in order to provide a reliable distribution service.
- *Congestion Control:* Fair competition with other streams, on shared links, is essential to avoid network congestion that would damage co-existing (e.g. unicast/email) services. This is not an issue for such fully-provisioned-multicast wireless networks as 3G MBMS or DVB-H systems, but is critical with content distribution over the public Internet.

Depending on the target use case, the above challenges may be met. Security is another challenging requirement in some situations, either to provide source authentication, message integrity, confidentiality and content access control, but it will not be detailed in the present thesis.

1.1.2 Delivery Service Models

We can distinguish three delivery models [45]:

- The *streaming* service model is typically used for audio and video content produced in real-time, but other kinds of content (e.g. the data provided in real-time by a probe or the subtitle of a movie) can be streamed too. Here timeliness is often more important than full transmission reliability.
- The *on-demand* service model, shown in figure 1.1, is typically used for the distribution of popular content. The content, which is not necessarily static but may change over time, is continuously broadcast or multicast, and interested clients join the session, download the content and leave the session whenever they want. Transmissions can be performed cyclically using a file carousel, and are not necessarily sequential. Indeed packets may be sent in a random manner to provide loss-pattern and session-joining-time independent performance to receiver. The service is reliable, scalable, but is generally not real-time constrained.

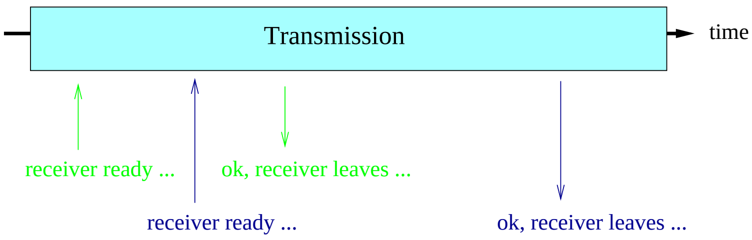


Figure 1.1: On-demand service delivery model.

- Finally, the *push* service model, depicted in figure 1.2, is a synchronous model, where all receivers are supposed to be ready before the transmission starts. It is a sender initiated model where the content is delivered from time t_0 for a finite duration. This model is typically used to deliver content to a selected set of receivers, and mechanisms using session announcements and receiver reception reports can be used to ensure a minimal synchronization between sender and receivers.

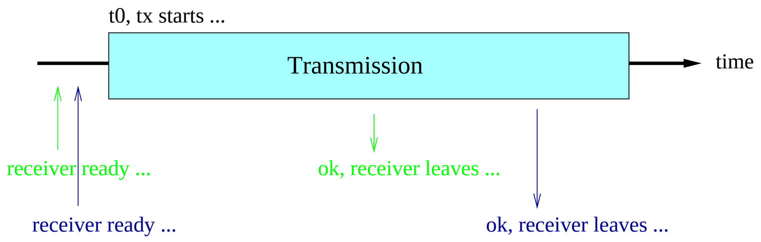


Figure 1.2: Push service delivery model.

1.2 RMT IETF Solutions

A "one-size-fits-all" protocol cannot comply with all the challenges and delivery models presented before. Therefore the Reliable Multicast Transport (RMT) IETF Working Group has adopted a modular approach [105]:

- Building Blocks (BB) are the basic components, (re)usable in different contexts and designed with flexibility in mind. They can be "plugged" or "unplugged" in order to enable or disable features according to the needs of a Protocol Instantiation (see below). The Forward Error Correction Building Block (FEC BB) is a particularly useful BB.

- A Protocol Instantiation (PI) is a set of BBs plus some PI-specific functions and headers (that are defined in a dedicated BB, for instance the Layered Coding Transport (LCT) BB for the Asynchronous Layered Coding (ALC) PI). Each PI aims to answer a very specific (or small set of) use case(s).

Thanks to this logical view, two PIs have been designed[1]:

- Asynchronous Layered Coding (ALC) [45]

- NACK Oriented Reliable Multicast (NORM) [6]

IP is a natural convergence layer on top of any physical layer, and so these RMT protocols rely on UDP/IP and are used either in a multipoint (their initial goal) or point-to-point scenario (which is sometimes useful). A general overview of these solutions is given in figure 1.3.

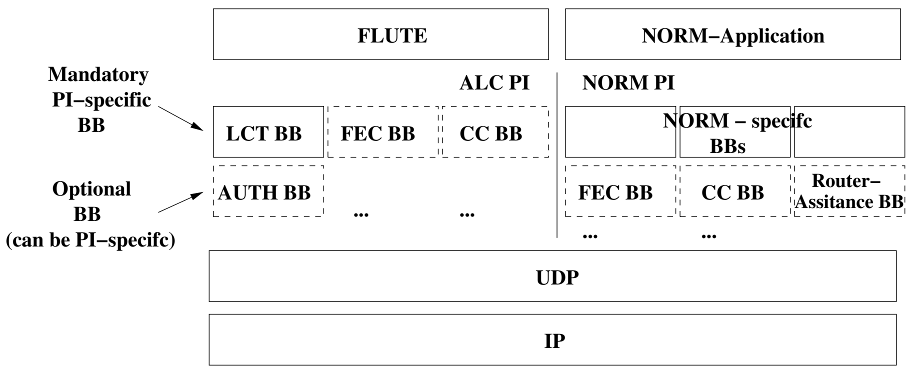


Figure 1.3: The RMT IETF solutions.

[1] A third PI, router assisted protocol Tree-based Acknowledgment/Generic Router Assist (TRACK/GRA), has been considered for some time, but removed from the scope of IETF RMT due to lack of progress.

1.3 Scope of this Thesis and Summary of its Main Contributions

The scope of this thesis is reliable multicast with massive scalability as a core requirement. The work clearly starts from the RMT IETF solutions, with a focus on Asynchronous Layered Coding with its building blocks and applications. Since NACK Oriented Reliable Multicast is not massively scalable, it is only considered to a lesser extend.

1.3.1 Contributions to Large Scale Content Distribution Building Blocks

Major contributions come from the investigation of the building blocks required in this context. Specifically the Forward Error Correction (FEC) building block is of particular interest in this thesis.

We identified the broader and well-known class of "Low Density Parity Check" (LDPC) large block FEC codes to have a very high potential. Therefore all our work turned around this class of codes, and we designed and analyzed derivates, in particular LDGM-Triangle, which yields very promising performances.

We designed a software implementation that implements these highly efficient codes. During the codec design, we put a lot of efforts on optimizing both the encoding/decoding speeds and the maximum memory requirements. Our results show that encoding and decoding are extremely efficient, even on small devices, and memory requirements acceptable. These results open its use to a wide field of applications and scenarios, including high speed networks and/or constrained devices (e.g. DVB-H compatible devices). We therefore believe that our codec offers an interesting and competitive alternative for many applications requiring an open-source, large block FEC codec.

The performance of the FEC codes largely vary, depending in particular on the code used and on the object size, and these parameters have already been studied in detail by the community. However the FEC performances are also largely dependent on the packet scheduling used during transmission and on the loss pattern introduced by the channel. Little attention has been devoted to these aspects so far. Therefore we analyzed their impacts on the three FEC codes: LDGM Staircase, LDGM Triangle, two large block FEC codes, and Reed-Solomon, a small block code. Thanks to this analysis, we defined several recommendations on how to best use these codes, depending on the test case and on the channel, which turns out to be of utmost importance.

Starting from this analysis a detailed performance study of four large block FEC codes, namely LDPC, LDGM, LDGM Staircase and LDGM Triangle, compared with a small block Reed-Solomon codec which serves as a reference has been made. The comparison focuses on three major aspects: the global decoding inefficiency (caused either by the FEC code or by the need to split the object into several blocks), the encoding and decoding speeds, and the memory requirements at the encoder and decoder. From these performance results, we identify the situations where they best operate.

Special attention has been put on multimedia delivery: The fully reliable delivery of multimedia content is not always guaranteed. A client does not necessarily receive a given media content entirely. Mechanisms like error concealment and error resilient video coding allow the receiver to deal with partially received data and to play the content, with decreased quality though. Packet-level Forward Error Correction is used as a complementary technique to counter the effects of losses and reconstruct the missing packets. However if the number of packets received is too low for the FEC decoding process to finish, the received parity packets may turn out to be useless, and finally more source packets may be unavailable to the application than if FEC had not been used at all. We analyzed the adequacy of the LDGM Staircase, LDGM Triangle and RSE FEC codes to offer a partial reliability service for media content distribution over any kind of packet erasure channel, that is to say a service that enables a receiver to reconstruct parts of the content even if the FEC decoding process has not finished.

Finally at building block level, we contributed with the analysis of the impacts of the startup behavior of several Multilayered Multicast Congestion Control Protocols on the performance of multicast content delivery protocols through mathematical models. Our results show (1) that these congestion control protocols have a major impact on the download time, and (2) that the three protocols considered in this study yield significant differences.

1.3.2 Contributions to Large Scale Content Distribution Applications

Our contributions detailed in the previous part helped us to investigate several applications:
We proposed a logical and physical file aggregation scheme for the file delivery protocol/application FLUTE. The logical file aggregation mechanism is a generalized grouping mechanism, allowing to logically group files. The physical file aggregation scheme allows, additionally to a logical grouping, to more efficiently use FEC in the context of FLUTE, in particular when dealing with a large number of "small" files. Unlike a solution based on the creation of an archive, the object aggregation scheme (1) avoids the need to perform preliminary transformations on the content and (2) preserves the possibility to extract a subset of the content, which may be a critical aspect with some partially reliable broadcasting test cases.
We also looked at large scale video delivery: Designing a video streaming solution for an environment having no or a constrained back channel due for instance to scalability issues leads to many challenges. We introduce SVSoA version 1 and version 2, two approaches that use a transmission technique close to the file broadcast methods we tackle in this thesis. We make no assumption on the target environment either, which can be either the Internet (if/when multicast routing is available), a site (e.g. a campus or an hotel), or a dedicated broadcasting network (e.g. a cable or DVB network). SVSoA is based on ALC, and can be used in all these situations. This approach emphasizes the reliability of transmissions in front of severe packet loss conditions and the stability of the video quality reconstructed.
Finally, we contributed to large parts of MCLv3/FLUTE library where we especially implemented the FLUTE file delivery protocol.

1.3.3 Standardization Activities

We conducted some standardization activities at IETF, in particular to standardize the LDPC Staircase and LDPC Triangle codes. There is a strong incentive at the RMT working group to have several FEC codes standardized. Our LDPC codes have therefore a strong probability to become one of these standardized codes. In parallel we proposed the LDGM-Staircase code at DVB-H, with the help of STMicroelectronics.
The logical and physical file aggregation scheme for FLUTE has also been presented at IETF. Discussions on this proposal is still in progress.

1.3.4 Other Contributions

During my visit at the INRIA I also contributed to other works, that are however out of scope of this document:

- Contributions to the *implementation of Robust Header Compression (RoHC)* [11] in the context of an industrial contract with STMicroelectronics.
- Implementation of and contributions to the *Opportunistic Encryption scheme* [16]: This lead to a publication [16] and an implementation [68] of the scheme.

1.4 Road Map of this Document

Our work is divided into three parts:

- Part I, *Overview of Large Scale Content Distribution Techniques*: Chapter 2 introduces Forward Error Correction Codes, a particular important building block in the context of our work. Next, Chapter 3 describes the Asynchronous Layered Coding protocol, the basic protocol on which almost all our contributions rely. The alternative, but less useful approach NACK Oriented Reliable Multicast is described in Chapter 4. FLUTE, a file delivery protocol/application build on top of Asynchronous Layered Coding, and a concrete usecase, namely MBMS in 3G cellular systems, are described in Chapter 5 and Chapter 6 respectively.
- Part II, *Contributions to Large Scale Content Distribution Building Blocks*: This part begins with the description of our contribution in the area of Forward Error Correction Codes in Chapter 7

to Chapter 11. More precisely Chapter 7 introduces the LDPC Forward Error Correction codes used and designed in this thesis. Chapter 8 describes the implementation details of these codes. The impact of packet scheduling and packet losses on Forward Error Correction performance is detailed in Chapter 9. Based on these first results, a more precise performance analysis and comparision is done in Chapter 10. The adequacy of Forward Error Correction Schemes for the usage with partially reliable media broadcasting schemes is analyzed in Chapter 11.

The startup behavior of multilayered multicast congestion control protocols is analyzed in Chapter 12.

- Part III, *Contributions to Large Scale Content Distribution Applications*: Chapter 13 describes our proposal for a file aggregation scheme for the FLUTE file delivery application. The MCLv3/-FLUTE library is also described in Chapter 13. The scalable video streaming approach based on Asynchronous Layered Coding is presented and analyzed in Chapter 14.

Last, Chapter 15 concludes this thesis. We should note that some parts of the thesis were published and are known to the community, while others are still being developed and opened the way for many future research topics.

Part I

An Overview of Large Scale Content Distribution Techniques

Chapter 2

Forward Error Correction Codes

Contents

2.1 Introduction

2.1.1 Principles

In a best-effort network, such as the Internet, the successful delivery of a packet to its receivers is not guaranteed. Only an acknowledgment-based protocol like TCP, that request retransmission of a lost packet until successful reception of it, can guarantee the fully reliable delivery of a content. However TCP is not appropriate for large scale content distribution and UDP is used instead. UDP does not offer any reliability and to compensate for packet losses two approaches exist: (1) An Automatic Repeat Request (ARQ) mechanism above UDP, where lost packets are retransmitted, and (2) Forward Error Correction (FEC), which transmits redundant data (called parity data) along with the original data, and allows to recover up to a certain number of possible losses at the receiver. A combination of these two approaches is also possible.

Binary versus Packet, Symmetric versus Erasure Channels

FEC is therefore a *key component* of content distribution techniques [79]. Conversely to the FEC codes generally described in information theory, the FEC codes are used at transport/application level, i.e. the codes operate on packets and not on bit sequences. Packets either arrive perfectly or are lost, and there is no need to deal with altered bit sequences or corrupted packets, since we assume that the packet checksum (CRC) corrected altered packets or, if this operation was not successful, completely dropped the packet.
In more general terms, FEC codes can operate both over Binary Symmetric Channels (BSC) and Binary Erasure Channels (BEC). With a BSC channel a bit arrives at the destination either perfectly or erroneously. It typically happens over noisy channels. On the opposite, in a BEC channel a bit either arrives perfectly or is lost in the channel.

A typical example of erasure channel, which does not operate on bit streams but on packet streams, as the Internet, and since a packer either arrives perfectly or is lost we talk about a "Packet Erasure Channel" (PEC).
Most FEC codes can operate on both types of channels (symmetric and erasure), but it is much simpler with a Packet (resp. Bit) Erasure Channels (PEC) (resp. BEC). This is normal since a PEC carries more information than a BSC (i.e. the packets received are error free).
In this work we only consider codes for the PEC, that complements any FEC code that may be used in parallel to protect bit streams over the physical link.

A small example

The basic idea of a FEC code can be explained with a small example (see figure 2.1): Assume that we want to transmit $k = 4$ packets over the network. The FEC code produces $n - k = 3$ additional parity packets, and transmit all packets (original source packets and the created parity packets) over the network. The receiver can now reconstruct the content if he receives *any* $k = 4$ packets out of the $n = 7$.

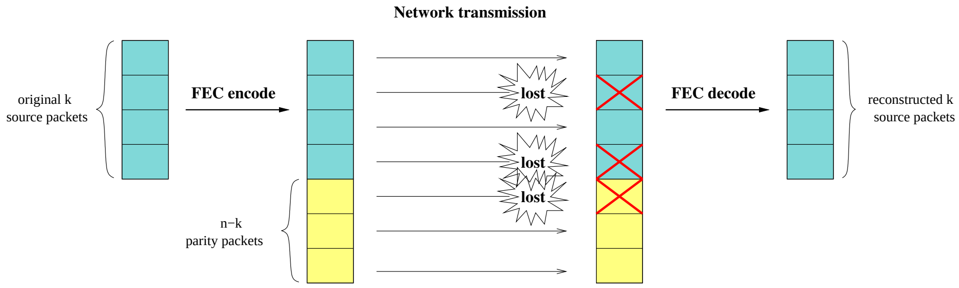


Figure 2.1: Example of a (7,4) systematic Forward Error Correction code.

General FEC Terminology

More generally, with a $(n; k)$ FEC block code, the k *source symbols* of a block are encoded into n *symbols* (i.e. encoded symbols).
A *symbol* is the basic and smallest entity of FEC codes which the basic operators (e.g. XOR) manipulate. With binary channels one symbol is a single bit, with packet channels a symbol is in most use cases an entire packet (as in the previous example), but it is also possible to operate on symbols smaller than one packet size.
With "systematic codes", the n encoding symbols are composed of the k source symbols plus $n - k$ additional *parity symbols* (A.K.A. repair symbols). We will use the terms repair symbol and parity symbol indifferently in this work. For non-systematic codes the n encoded symbols are all different from the source symbols. However we do not consider non-systematic codes in this work.
A receiver can then reconstruct the k source symbols on condition it receives any k symbols out of the total n (or a few percents more than k symbols with expandable and large block codes). The great advantage of using FEC is that the same parity symbols can be used to recover different lost symbols at different receivers.

Block Codes versus Convolutional Codes

Block codes work on fixed-size blocks of bits or symbols of predetermined block size k (k being the number of symbols per block), while convolutional codes work on bit or symbol streams of arbitrary length. Convolutional codes operate on serial data, one or a few symbols at a time. Block codes operate on relatively large (the size largely depends on the class of codes used) blocks.
Typical convolutional codes are based on the Viterbi algorithm where the encoded symbols depend not only on the current input symbols, but also on all past inputs. This concept has been refined, extended and highly optimized which lead to Turbo codes.

In this work we only consider block codes and we won't further detail convolutional codes. Interested readers can refer to [103, 28, 13]

2.1.2 FEC Fields of Application

Use of FEC for Reliable Content Distribution

If large block FEC codes have a natural field of application with the reliable multicast-based distribution of content to a large set of receivers over the Internet (e.g. using the FLUTE/ALC application), this is not their only field of applications: media broadcasting to vehicles [25], IP datacasting over DVB-T/H [26, 81] or MBMS in 3GPP [4] are applications that require the broadcasting of large contents to either a single client, or at the other extreme, to a huge number of clients that enjoy a unidirectional connectivity with perhaps large variations of quality (e.g. when a vehicle enters a tunnel or is behind obstacles). The use of large block FEC codes is therefore a natural solution to counter these long-term losses.

Use of FEC in Environments with Intermittent Connectivity

Several environments are characterized by an intermittent connectivity. [25] provides an example which nicely fits in this category: the Land Mobile Satellite (LMS) channel. The goal of the application is to distribute multimedia contents to the car passengers from a satellite connection. In this system, the mobile (car) experiences both fast fading and white Gaussian noise, which is addressed by the DVB-S error correction schemes, and slow fading (e.g. when entering a tunnel), which is addressed by a packet level FEC encoding. This two level FEC encoding is fully complementary, each level of protection addressing different problems. As mentioned in [25], the large block FEC codes that we present in this work are well suited to this kind of environment.

Use of FEC for Distributed Storage

FEC also proves to be of great help in distributed storage systems, where the object is replicated across a set of servers. Here clients download the object by accessing multiple servers in parallel. Such replicated systems improve both fault-tolerance and performance over non replicated storage systems. Using an FEC erasure code to dilute the information over the storage servers allows a greater and more flexible choice to the clients, and can enhance the overall downloading throughput. Basically, for each of the B blocks of a file, FEC encoding produces $n = fec\_ratio * k$ symbols, where the FEC expansion ratio is an integer ≥ 1. The system can now replicate any of the $B * fec\_ratio$ "encoding blocks" on the servers, and downloading any B different "encoding blocks" out of $B * fec\_ratio$ possibilities is sufficient to reconstruct the original file.
Several such systems, relying on various kinds of FEC codes have been proposed, like [37, 21, 64, 88] to cite only a few.

2.2 Simple FEC Codes

There are some very simple codes that are effective for repairing packet loss under very low loss conditions.

Simple XOR code

The certainly simplest FEC code is the XOR FEC code [58, 84], that provides protection from only a single loss. Only a single parity symbol is created, which is equal to the XOR sum of all source symbols. The XOR FEC code is therefore always a $(k + 1; k)$ code, where k is the number of source symbols.
For example if the source block consists of four source symbols s_0, s_1, s_2 and s_3 the value of the parity symbol p_4 is $p_4 = s_1 \oplus s_2 \oplus s_3 \oplus s_4$.
In this example, any single source symbol of the source block can be recovered as the XOR sum of all the other symbols. For example, if symbols s_0, s_1, s_3 and p_4 have been received, the missing source symbol s_3 can be recovered by calculating $s_2 = s_0 \oplus s_1 \oplus s_3 \oplus p_4$.

A slightly more sophisticated way of using the simple XOR codes is to segment the content into several blocks, and to apply the simple XOR code on each block. The number of parity symbols created is a bit higher but error protection of one parity symbol is limited to its block.

2.3 Small Block FEC Codes

Small block FEC codes are characterized by the fact that they are *only efficient on small source block sizes*, i.e. when k and n are relatively small. With higher k and n encoding or decoding times becomes prohibitive.

Reed-Solomon

The most used small block FEC codes are certainly the Reed-Solomon erasure codes (RSE). Two versions of RSE exist: one based on Vandermonde matrices [91] and the second less used based on Cauchy matrices [39].
RSE is intrinsically limited by the Galois Field (GF) it uses [91], so optimal k and n are relatively small. A typical example is GF(2^8) where $k \leq n \leq 256$. With one kilobyte symbols, a FEC codec producing as many parity symbols as data symbols ($n = 2k$) operates at most on 128 kilobyte blocks. All files exceeding this threshold must be segmented into several blocks, which reduces the erasure protection.
Using GF(16) solves this problem since here $n \leq 65536$. But a major drawback then is a huge encoding/decoding time. [91] reports encoding times in $O(k(n-k))$, and decoding times in $O(kl)$, where l is the amount of missing symbols, plus an additional cost of matrix inversion in $O(kl^2)$ (the inversion is done once for each block). This is the reason why for Reed-Solomon GF(2^8) is often preferred to GF(2^{16}) in spite of block size limitations.

The coupon collector problem and MDS codes

More generally with small block erasure codes objects exceeding the maximum block size k, have to be segmented into several blocks. If B blocks are needed, a symbol chosen randomly out of all blocks has a probability $1/B$ to recover a given erasure. $B = 1$ is clearly the optimal solution, and therefore segmenting an object into several block clearly decreases erasure protection performance. This phenomenon is known as the “Coupon Collector Problem” [15].
Yet if considering only a single block, small block codes are optimal because a receiver can perform FEC decoding as soon as it has received *exactly* k symbols out of the n. Such codes are called ”Minimum Distance Separation” (MDS).

2.4 Large Block FEC Codes

Conversely *large block codes* support large k values (several hundreds of thousands symbols and more) while keeping *high performance in encoding and decoding.*
The “Coupon Collector Problem” is completely suppressed when a single block is used (which is often feasible), and it is largely reduced if several blocks must nonetheless be defined (e.g. because of practical block size limitations with a given codec).
They are however not MDS, i.e. $(1+\epsilon)k$ symbols are required at a decoder for the decoding of a block of size k to complete. ϵ is generally quite small, and can be experimentally evaluated (see chapter 10). Even though large block FEC codes are not MDS, the overall reception overhead is usually smaller than that of RSE with large objects, while being an order of magnitude faster.

LDPC

Most codes derive from the well known Low Density Parity Check (LDPC) codes introduced in the 1960s [31]. LDPC codes rely on a parity check matrix, H, which forms a system of linear equations between source symbols, s_i, $i \in \{0..k-1\}$ and parity symbols: p_i, $i \in \{k..n-1\}$. The basic operator is `XOR`. A simple example is shown in figure 2.2.

$$
[\, H_1 \,|\, SC_5] = \begin{array}{c}
\begin{array}{cccccc|ccccc} s_0 & & \cdots & & & s_5 & p_6 & & \cdots & & p_{10} \end{array} \\
\left(\begin{array}{cccccc|ccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\end{array}
\qquad
\begin{array}{l}
s_1 + s_3 + s_4 + s_5 + p_6 = 0 \\
s_0 + s_1 + s_2 + s_5 + p_6 + p_7 = 0 \\
\qquad \cdots
\end{array}
$$

Figure 2.2: Parity Check Matrix H of an (6;11) LDPC-Staircase code.

If the system is built appropriately, encoding is extremely fast and can be done in $O(n-k)$. One cannot know in advance how many symbols must be received before decoding is successful (LDPC codes are not MDS), so decoding is performed progressively after each packet arrival. Due to the simplicity of LDPC, the decoding process (A.K.A. iterative decoding) is extremely fast too.
The LDPC-Staircase and LDPC-Triangle are two variants detailed in [97] and for which a publicly available GNU/LGPL code implementations exist [77].
More details on LDPC codes and especially its variants LDPC-Staircase and LDPC-Triangle, as well as precise performance evaluations and details on its implementation are given in Chapter 7 and following chapters.

Tornado

Tornado codes [56, 55, 54, 53] are also a variant of LDPC codes, however they are heavily patented and only commercial implementations exist.
The idea is to construct several levels of parity symbols. A first level that is created exclusively from source symbols, a second level of parity symbols that is generated using only the parity symbols of the first level and so on. This can be represented with several cascaded matrices as it is depicted in figure 2.3. The number of parity symbol levels used and the degree distribution (i.e. the number of ones per column and line) follows an appropriate (and patented) rule described in [52].

First level of parity symbols

$$
\begin{array}{c}
\begin{array}{cccccc|ccccc} s_0 & & \cdots & & & s_5 & p_6 & & \cdots & & p_{10} \end{array} \\
\left(\begin{array}{cccccc|ccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
\end{array}
$$

Second level of parity symbols

$$
\begin{array}{c}
\begin{array}{ccc} & & p_{11} .. p_{13} \end{array} \\
\left(\begin{array}{ccccc|ccc}
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}\right)
\end{array}
$$

Third level of parity symbols ...

...

Figure 2.3: Principle of Tornado codes.

2.5 Expandable/Rateless FEC Codes

With large block codes, the n value must be defined before encoding is performed, and cannot be changed afterward (since it defines the H matrix size). Moreover when a high number of encoding symbols are produced, the codes efficiency is reduced. Expandable codes do not have these limitations. They can operate on very large blocks (k) and an unlimited number of parity symbols ($n-k$) can be produced without affecting the erasure recovery capabilities. As the "code rate" (k/n ratio) can be very small, they are also called "rateless codes".

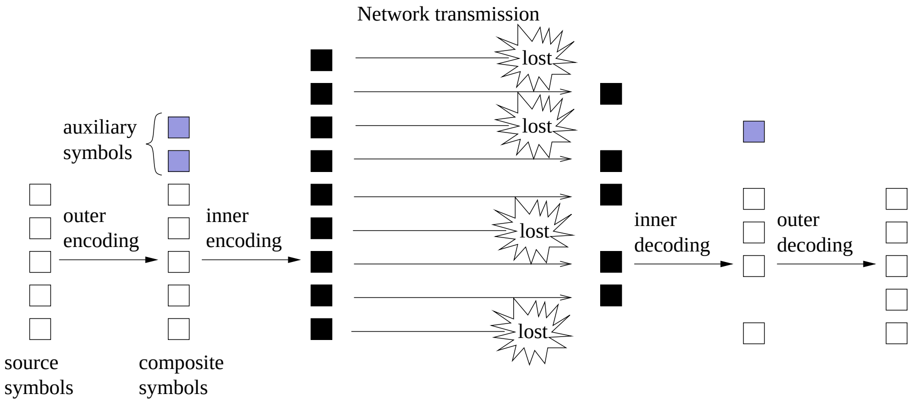


Figure 2.4: Overall design of online codes.

Expandable codes are well suited to content broadcasting since the sender often wants to produce new parity symbols on the fly, without limitations (e.g. in on-demand mode). With these codes, a receiver is guaranteed not to have duplicated parity symbols.
LT [44] and Raptor [100] codes fall in this category. They are heavily patented and only commercial implementations exist. Online codes [62, 63] are also expandable codes, and conversely to LT and Raptor its creator have not claimed any patent on this code. However it is not clear if the code is really totally free of any intellectual property right, since it has become very difficult today to design an expandable FEC code that is not covered by the high number of existing patents in this domain. In spite of these uncertainties we quickly present Online codes in the following.

Online codes

Online codes do not rely on any parity check matrix in order perform decoding nor encoding. They are based on two levels of FEC encoding (see figure 2.4):

- A first outer encoding step where a small amount r (generally $r = 0.01 * k$) of "auxiliary" symbols are created. Auxiliary symbols are computed as the XOR sum of a number of source symbols: using a PRNG, for each source we chose q (generally $q = 3$) auxiliary symbols, to which the source symbol is summed. The k source symbols plus the r auxiliary symbols forms the "composite" symbols.

- A second inner encoding step, which is able to generate repair symbols on-demand based on the $k + r$ composite symbols. There is no predefined limit of generated repair symbols.

  Each repair symbol has an unique identifier (the authors suggest a 160 bit value for the identifier). The identifier is used to seed a PRNG, which will return a sequence of composite symbol indices. These symbols are XORed in order to generate the new parity symbol. The degree, i.e. the number of composite symbols used to generate each parity symbol, depends on a distribution function, i.e. when choosing a degree of an equation, this has to follow this distribution function (the exact function used is detailed in [62, 63]).

  More generally, it has been shown that using an irregular distribution of the degrees of an equation system, the forward error correcting capacities are much more interesting than when using a regular distribution (i.e. all equations have the same degree) [52].

An overall of the design of online codes is given in figure 2.4.

2.6 Conclusion

The purpose of this chapter is to only make a basic introduction to the several FEC classes and FEC codes that exists. The codes presented here yield very different behaviors regarding error correcting capabilities, encoding and decoding speeds and memory requirements, and not all codes are adapted to specific use cases. It would have been too complex to implement and precisely evaluate each of the codes presented. Moreover many patents in this area largely restrict the usage of some of the FEC codes.

Therefore in this thesis we focus on LDPC large block FEC codes, and precise evaluations as well as implementation details are given in Part II of this thesis. We also give several points of comparision to other FEC codes, but the level of detail for other codes will be less important than for LDPC and its variants.

Chapter 3

Asynchronous Layered Coding (ALC)

Contents

3.1 Introduction and Principles

The Big Picture

Asynchronous Layered Coding (ALC) is a content delivery protocol that does not require any feedback from the receivers to the sender. This makes it *massively scalable*, i.e. millions of receivers are easily supported. This feature also enables ALC to be used over unidirectional links where there is no (or a limited) return channel, as for instance in the context of satellite broadcast.
ALC has been designed to support the three delivery service models introduced before: *push*, *on-demand*, and to a lesser extent *streaming*. We will show in chapter 14 an alternative way of using ALC with a streaming delivery service model, e.g. for audio/video delivery.
As there is no feedback, no repeat request mechanism can be in-band with ALC that would enable the source to adapt its transmission according to the feedback information sent by the receiver(s). In order to provide a fully (or partially) reliable service, *ALC largely relies on the use of FEC*. After a FEC encoding of the content source blocks, redundant data is transmitted along with the original data in the systematic FEC case. Thanks to this redundancy, a certain number of lost packets can be recovered at the receiver. Another major asset of FEC is that the same parity symbol can potentially recover different losses at different receivers, which largely increases transmission efficiency.
ALC also meets the channel and client heterogeneity challenges. Concerning the channel heterogeneity, a theoretically infinitely long (e.g. in on-demand mode) ALC session that repeats symbol transmissions guarantees that clients can decode the content successfully, independently of all other receivers and no matter the number of losses and the loss patterns experienced. Therefore, ALC provides *high robustness*. ALC also supports receiver heterogeneity in terms of sustainable reception rate, by using (optional) *multi-layered transmission* (section 3.2.3). Each receiver independently chooses how many

layers to receive, according to its bandwidth and processing capacities. Therefore all receivers behave in a completely asynchronous manner.

Terminology

The following terminology will be used: ALC transports *objects* (e.g. files). Each object is logically divided into (one or more) *blocks* of appropriate size upon which FEC codes calculate source and parity symbols.

3.2 ALC Building Blocks

ALC is designed to be highly flexible. To do so ALC is composed of building blocks, that can be plugged or unplugged to enable or disable features according to the application needs.

3.2.1 Layered Coding Transport (LCT) Building Block

The LCT Building Block [47] defines the basic features and the LCT header that provides a number of fields to convey in-band session information to receivers.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   V   | C | r |S| O |H|T|R|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)  |
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Sender Current Time (SCT, if T = 1)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Expected Residual Time (ERT, if R = 1)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Header Extensions (if applicable)              |
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.1: Default LCT header format

An ALC/LCT session is fully identified by the $\{source\ address; TSI\}$ tuple, where TSI is the *Transport Session Identifier*. The destination IP address and the destination UDP port do not identify an ALC session, but rather, they are only used in transmission for distinguishing between ALC channels (layers) of a certain ALC session.
Each object is uniquely identified by its *Transport Object ID* (TOI). Nothing is specified on how to generate the TOI: it may be incremented for each new object, or be the hash of the object. For that reason, the size of the TOI field is not predefined ([47] defines a $[16; 112]$ bit range).
LCT defines a *Congestion Control Information* (CCI) field to associate a congestion control (CC) building block. Yet LCT does not specify the CCI field format which must be specified by the congestion control building block itself. The field is set to null if no congestion control is used (e.g. in 3G and DVB-H environments - which are not instances of the public Internet)
Finally LCT defines *Header-Extensions* to carry additional information. Optional header fields that are not always used or have a variable size can be added this way. For instance the FEC building block and the authentication building block heavily rely on these extension headers.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         UDP header                            |
|                                                               |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                     Default LCT header                        |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       FEC Payload ID                          |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Encoding Symbol(s)                        |
|                           ...                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.2: Overall ALC packet format

The LCT header is placed just after the UDP header as it is depicted in figure 3.2. The LCT header is followed by the FEC Payload ID (presented in the following) to identify the symbols transported.

3.2.2 FEC Building Block

The FEC building block [46][104] is a particularly important building block in achieving scalability and reliability with ALC (and NORM), even though it is not mandatory. However a FEC code alone is not usable, since the sender and receivers must: (1) identify the codec, (2) identify each symbol, and (3) specify the various parameters used by the codec for a given object. To that purpose the FEC building block defines:

1. FEC Encoding ID and FEC Instance ID: These two values fully identify the codec. The FEC Encoding ID is the primary identifier, and identifies either specific FEC codes with *fully-specified* schemes, or a class of codes with *under-specified* schemes. In the latter case, the FEC Instance ID must be used in order to completely identify the codec.

   Fully specified schemes are typically associated to very simple FEC codes (like the NULL code, i.e. no FEC, essentially used to test implementations) or for commonly used codes (such as Reed-Solomon, which should be part of the fully specified codes). The FEC Encoding ID range [0,127] has been reserved for fully specified codes.

   Under-specified schemes generally regroup a class of codes, as for instance the small block systematic codes that use the FEC Encoding ID 129 or the broad class of Low Density Parity Check (LDPC) codes that use the FEC Encoding ID 132 [98]. The FEC Instance ID is needed to further identify the code used. For instance FEC Encoding ID 132 and FEC Instance ID 0 identifies the LDPC-Staircase code.

2. FEC Payload ID (FPI): Each FEC Encoding ID is associated with a FEC Payload ID, which uniquely identifies each symbol of an object after FEC encoding. For instance with the under-specified FEC Encoding ID 128 [46], each encoding symbol of a given object is identified by its Source Block Number and Encoding Symbol ID as it is shown in figure 3.3.

   Other informations may be carried within the FPI depending on the FEC code used. For instance with an FEC Encoding ID 129 [46], especially used for Reed-Solomon, an additional parameter *Source Block Length* appears and allows to dynamically change the source block length for each block (as it is done in NORM (chapter 4)).

3. FEC Object Transmission Information (FEC OTI): To each FEC Encoding ID, a FEC OTI must be specified to synchronize the codecs at both ends with respect to the various parameters of an object (e.g. maximum source block length, object size, etc.).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Source Block Number                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Encoding Symbol ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.3: FEC Payload ID for an FEC Encoding ID 128

An FEC OTI example with an FEC Encoding ID 132

In the particular case of LDGM-Staircase codes, that are part of the broader class of Parity Check Matrix-based codes, we have identified in [98, 84]:

- the total length of the object in bytes called *Transfer Length*;
- the maximum number of source packets that can compose any block, *Max. Source Block Length*, or in short *max_k*. This field allows receivers to compute the source block structure of the object (section 3.2.2);
- the maximum number of encoding packets that can be generated for any source block, *Max. Number of Encoding Symbols*, or in short *max_n*. This field allows receivers to compute the number of parity packets of each block (section 3.2.2);
- the packet size, called *Encoding Symbol Length*;
- and the seed of the PRNG used by the FEC code for this object.

This information is sent within the ALC session, either within a dedicated header extension, `EXT_FTI`, shown in figure 3.4 that is appended to the ALC header, or if used within the file delivery application FLUTE build on top of ALC (detailed in chapter 5), as part of the File Delivery Table that describes each file properties, using an XML based format.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   HET = 64    |      HEL      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                        Transfer Length                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   FEC Instance ID             |    Encoding Symbol Length     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Maximum Source Block Length                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Maximum Number of Encoding Symbols             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                     Seed (of variable size)                   .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.4: FEC OTI for an FEC Encoding ID 132

The `HET` and `HEL` fields are respectively the Header Extension Type and Length, and `FEC Instance ID` is an identifier that (along with the FEC Encoding ID, not shown here) uniquely identifies the codec used.

Several Blocks are Sometimes Needed...

In practice, a single "huge" object may have to be segmented into several blocks, even if we are dealing with so-called large block FEC codes. Indeed, encoding and decoding often require that the whole object be kept in memory, which leads to practical limitations. The number of blocks is nevertheless several orders of magnitude smaller than with Reed-Solomon codes, where a block size is limited to a hundred KB, or a little bit more.
The direct consequence is that the sending and receiving applications must agree on an appropriate blocking structure for each object exchanged. Two algorithms are defined to this purpose:

1. **The Blocking Algorithm**

   Defining this blocking structure, i.e. the number blocks and their size, is the goal of a dedicated algorithm, specified in FLUTE [82]. Given max_k, the transfer length, and the packet size (taken from the FEC OTI), this algorithm defines the optimal blocking structure, with all blocks being the same length, except maybe a few ones (at the end) whose size will be one packet smaller than the others.

   The goal of making all blocks almost the same size is to avoid sub-optimal situations made possible by more trivial algorithms. For instance if all blocks (except the last one) are of maximum size, max_k, the last one may be significantly smaller. At an extreme case, it could be one packet long. The protection of the last block would then be rather low, thereby compromising the whole transfer efficiency. Indeed, in the extreme case, with a FEC expansion ratio of 2, a receiver would have to receive one of the two encoding packets of the last block, "lost" in a total of $2 * (object\_size\_in\_packets - 1)$ source plus parity packets.

   Note that in some cases the Blocking Algorithm is not necessary: with an FEC Encoding ID 129, especially used with NORM, each block can define its own block size, independently of the other blocks. The idea is to progressively cut the object into several blocks, with a block size that may change from one block to another. The block size is included in the FPI.

2. **The "n-Algorithm"**

   Another constraint when dealing with several blocks is to agree on the n value used by the encoder for each block. Indeed, the FEC OTI only carries the max_n value for a block of maximum size, not the n value used effectively for each block. This is the goal of the "n-algorithm" defined in [98, 84].

   At the sender, the max_n value is first computed, taking into account the desired FEC expansion ratio specified by the user, for a block of maximum size, max_k. The n value for any other block is then determined by a simple rule of 3, and rounded down to the nearest integer to avoid problems. At the receiver, the same calculation is made, taking into account the max_k and max_n values found in the FEC OTI, plus the output of the blocking algorithm. The receiver therefore knows the actual $k, n, seed$ values for all blocks, and can initialize the FEC codec appropriately for each of them. The whole process is summarized in Figure 3.5.

3.2.3 Congestion Control Building Block

When used over the public Internet, ALC must include a congestion control building block. In order to preserve the massive scalability, only feedback free schemes are defined. Therefore congestion control is multi-layered and completely receiver-driven: the content is multicast on several layers, each layer being associated to a distinct multicast group. Then each receiver chooses dynamically how many layers to receive, depending on the sustainable transmission rate along the path to this receiver.
Several (more or less) TCP-friendly protocols have been defined: RLC [101], FLID-SL/DL [14], [59], and WEBRC [49] [48]. The choice of the congestion building block may be dictated by the desire to have a standardized solution (WEBRC is an IETF RFC), or to minimize implementation complexity (RLC and FLID-SL are relatively simple, easy to implement building blocks, although they both have known limitations).
The client behavior will differ according to the protocol: transmissions will take place either at some fixed predefined rates on each layer (RLC, FLID-SL), or using a cyclic, dynamically changing bit-rate

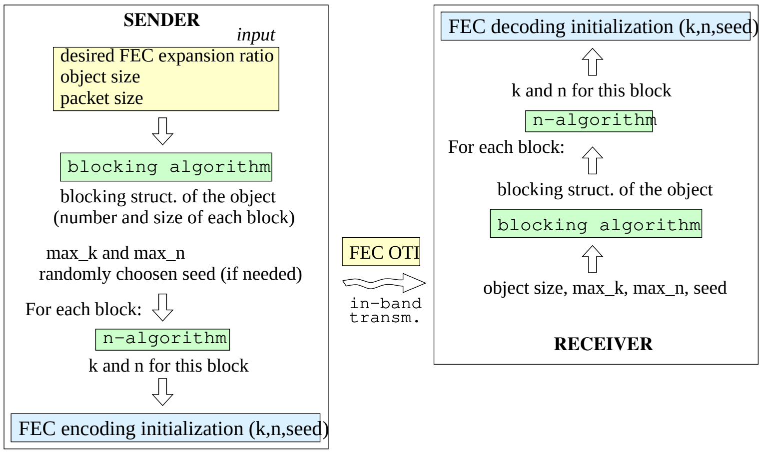


Figure 3.5: Steps required for synchronizing sender and receiver(s) for a given object.

(FLID-DL, WEBRC); the protocol can either be equation based (WEBRC) or event based (RLC, FLID). Naturally, these fundamental differences impact upon their performance and TCP-friendliness. When used over provisioned, pre-allocated, channels (e.g. within 3G MBMS and DVB-H environments), no transport layer congestion control is needed.
A detailed discussion on the behavior and the impacts of the congestion control protocols RLC, FLID-SL and WEBRC is given in Chapter 12.

3.2.4 Authentication Building Block

The goal of this building block is to perform both a packet source authentication and packet integrity check. The TESLA authentication approach [87, 86] is a good solution that is well suited to the case of high rate transmissions over lossy channels. [27] gives more details on how to use it in an ALC (or NORM) context.

3.2.5 Session Description Building Block

Finally, the various ALC session parameters (e.g. congestion control protocol, sender's IP address, TSI, etc.) must be communicated to the receivers. If ALC does not explain how to do that, several solutions exist: Session Description Protocol (SDP) [33], XML metadata [66], or other out-band mechanisms.

3.3 Fields of Application

ALC has *a very large field of application*, much larger than the protocol NORM for instance (presented in the following chapter). ALC keeps the same effectiveness regardingless the number of receivers, using unicast or multicast, thanks to the suppression of any feedback from the receivers to the source, and all receivers can act completely asynchronously.
ALC can be used on any type of medium, an unidirectional one, or, more classically, a bi-directional one, and its robustness support merely all types of packet losses, even the extrem case when a channel is temporarily not available (e.g. when a car crosses a tunnel). Therefore ALC can be used for instance on unidirectional satellite links or for the distribution of content to cell phones or vehicles. It may also be used in networks like a LAN or the Internet since congestion control is a supported feature.
The application build on top of ALC can either be used for file transmissions with an application such as FLUTE (presented in chapter 5) in on-demand or push mode. FLUTE may also be used for service announcements purposes, as it is done in the context of MBMS [4]. ALC may also be used for the streaming of video for instance (in streaming mode) with an application such as SVSoA (presented in Chapter 14). In the case of streaming the differentiation between slow and fast receivers is done on

the amount of information received (and decoded, if FEC is used), resulting in a differentiated video quality. For content distribution in on-demand or push mode when used with FLUTE for instance, the differentiation between receivers is only done one the time of reception required by each client.

Chapter 4

NACK Oriented Reliable Multicast (NORM)

Contents

4.1 Introduction and Principles

The NORM approach essentially relies on retransmission requests (Negative Acknowledgments or NACKs), and optionally on positive acknowledgments (ACKs) after an explicit request from the source. Due to this feedback mechanism, NORM scalability is limited to *small or medium* sized groups [7]. Large groups are a problem because the timer-based feedback suppression mechanism (used to increase NORM scalability) is still not sufficient to prevent feedback implosion in this case. Receivers should be *relatively homogeneous* and have comparable and sustainable reception rates. Indeed the associated congestion control protocol adapts the transmission rate to the slowest receiver. Consequently, a small number of receivers can easily slow down an entire NORM session. This is not scalable even under the best assumptions, since the throughput goes down with $1/\sqrt{number\ of\ receivers}$ [17].
NORM is a more *complex protocol* than ALC. For instance there are 13 different packet types! We will not detail NORM as we did for ALC because this protocol is less interesting in our context. Interested readers can refer to [6]. We only give some general ideas.

4.2 NORM Building Blocks

[5] defines several building blocks, in particular:

- NORM Sender Transmission strategies
- NORM Repair Process with timer-based feedback suppression
- NORM Receiver Join Policies

Additional building blocks can also be plugged into NORM:

- The FEC building block [46]: With FEC Encoding ID 129, which is often the preferred scheme, the block size can be adapted dynamically according to the feedback received. With a high loss rate, the sender may choose to reduce the block size in order to be able to generate a higher number of parity packets and vice-versa.
- The Round-Trip Timing Collection building block and the Group Size Determination/Estimation building block: The RTT and group size estimates are then used by other building blocks (e.g. the congestion control building block).
- The congestion control building block: One of two building blocks may be used: PGMCC [92] and TFMCC [106]. Both of them rely on the round-trip timing collection building block.
- The authentication building block: Here also the TESLA authentication scheme can be used, as explained in [27].
- The Router/Intermediate System Assistance building block: This building block turns NORM into a router assisted protocol (similarly to TRACK/GRA), for instance by supporting feedback suppression and/or aggregation mechanisms, in order to increase NORM scalability.

Other building blocks are specified in [5], but we will not detail them. Nevertheless the number building blocks defined once again emphasizes the complexity of the protocol.

4.3 Principal NORM Packet Types

NORM uses a high number of packet types and we will detail the most important here:

- NORM_DATA: NORM_DATA packets carry the actual data. The data may be source or parity symbols.
- NORM_NACK: NORM_NACK packets are sent by the receivers to request retransmission of lost packets. NACK packets are only sent if explicitly requested by the source with an FLUSH command, which is normally sent when the source transmits the last source symbol of a block.

  NACKs are send using a probabilistic feedback supression mechanism. Each receivers randomly choose a *backoff timer*. The NACK is only transmitted if at the end of the backoff timer no identical NACK has been observed (possibly sent another receiver) and if the loss hasn't been corrected. The maximum value of the backoff timer is dependent on the estimated group size and the maximum RTT value between the source and each receiver.

  The format of NORM_NACK packets is very flexible: the receiver can explicitly list each missing symbol, give a range of symbol identfiers or specifie the number of requested parity symbols. It is also possible to specify missing blocks or a range of missing blocks. A single NACK packet is therefore able to signal several losses, and this limits the number of messaged that needs to be sent. On the other side processing at the source is more complex.
- NORM_CMD: command packets are sent from the source and may be used for different purposes:
  - NORM_CMD(FLUSH): the FLUSH command is an explicit request to the sender to send retransmission requests (NACK) if needed.
  - NORM_CMD(ACK_REQ): the ACK_REQ explicilty requests a specified set of receivers to acknowledge the reception of packets within a given range, or to request retransmission of lost packets (the range and the set of receivers are specified in the ACK_REQ message).
  - NORM_CMD(SQUELCH): the SQUELCH command informs the receivers on the current valid retransmission window. This command is typically used if a retransmission request outside the current retransmission window is received by the source. It allows to resynchronize the receivers.
  - NORM_CMD(EOT): the end of transmission command (EOT) indicates the end of the NORM session to the receivers.
  - NORM_CMD(CC): the CC command is specific to the congestion control building block used.

4.4 NORM versus ALC

We now compare NORM and ALC with respect to the challenges introduced in section 1.1:

- *Scalability:* Only ALC supports massive scalability;
- *Channel and Client Heterogeneity:* Only ALC supports client heterogeneity.
- *Content Heterogeneity:* Both protocols support file transmission, but NORM is not suited when data has real time constraints.
- *Reliability:* For long reception blackouts, ALC offers more robustness than NORM;
- *Congestion Control:* Both protocols can use congestion control.

Therefore, in the following we focus on ALC since it better addresses the challenges introduced earlier, offers a higher flexibility, can be adapted to very different contexts, and is less complex.

Chapter 5

FLUTE for File Delivery

Contents

5.1 Introduction and Principles

So far, we described the basic protocols and building blocks that are available for reliable distribution of bulk data. However, the ALC protocol itself is underspecified (as well as NORM) and so not sufficient for file delivery, since the file properties must be delivered too.
File Delivery over Unidirectional Transport (FLUTE) [82] is a fully-specified file delivery protocol/-application *built on top of ALC*. It inherits all the major assets of ALC: an unlimited scalability, the ability to use almost any IP transmission channel (unidirectional or not), and a high robustness. FLUTE transmits meta information for each file delivered in a file delivery session.The meta-information contains such information as the file identifier (as a URI which may include the file name), file size, content type (MIME type) and content encoding (such as compression). It is capable of delivering, for instance, the required video codec and textual description along with the video media file itself. A receiver can then choose to decode all or a subset of files based on this information. For example, a receiver without a certain video codec may chose to drop all ALC packets associated with files using that video codec (to avoid processing data which could finally be unusable).

5.2 File Delivery Table (FDT)

The meta-information of *all files* of a FLUTE session is contained in the session's abstract File Delivery Table (FDT). To each file is associated at least a *TOI* (used by ALC to identify the object) and a file URI (Uniform Resource Identifier) (*Content-Location* attribute). Additional information may be specified, including: the file size (*Content-Length*), the MIME-type (*Content-Type*), and a MD5 hash (*Content-MD5*). The FDT information is not generally static, in particular an attribute may change (e.g. a file is renamed), and files may be added or dropped dynamically.
The FDT is delivered via *FDT-Instances*, which are discrete XML representations of a subset of the abstract FDT. To represent its dynamic nature, each FDT-Instance includes an expiry timestamp (*Expires*), which applies to all data contained in that FDT Instance. Additionally an FDT-Instance may contain the *Complete* attribute to indicate that the meta-information transmitted so far will not

change anymore. Once used, there can be no file addition or attribute change and no additional files will be added to the session - making it simpler for a receiver to determine when it has received all interesting files for a particular session.
FDT-Instances can also be used to transmit FEC OTI informations, i.e. all the parameters used by the FEC code, instead of using the header extension `EXT_FTI` (see section 3.2.2). To this purpose attributes such as *FEC-OTI-FEC-Encoding-ID*, *FEC-OTI-FEC-Instance-ID*, *FEC-OTI-FEC-Maximum-Source-Block-Length*, to cite only a few, can be specified within a FDT-Instance.
It is the sender's role to decide when and how to transmit FDT-Instances (i.e. which subset of the known valid FDT information to include in FDT-Instance). It is, however, recommended to transmit a file description before the file itself, so that the receivers know how to handle incoming ALC packets, e.g. by dropping them, or directly writing on disk, or keeping them in memory. There are several ways to schedule the FDT-Instances. In on-demand mode, they must be periodically transmitted to support mid-session receiver-joins - enabling receiver selection and download of content they are interested in, at any time. More information on how to use FLUTE regarding object and FDT-Instance scheduling is given in section 5.3.1.
FDT-Instances are transmitted with the *dedicated TOI value 0*. By searching the TOI of a received packet in its cached FDT database, a receiver easily finds the associated file and can decide to process it or drop it.

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:fl="http://www.example.com/flute"
xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
Expires="2890842807">
  <File  Content-Location="http://www.example.com/menu/tracklist.html"
         TOI="1"
         Content-Length="6100"
         Content-Type="text/html"
         Content-Encoding="gzip"
         Content-MD5="+VP5IrWploFkZWc11iLDdA=="/>
  <File  Content-Location= "http://www.example.com/tracks/track1.mp3"
         TOI="2"
         Content-Length="392103"
         Content-Type="audio/mp3"
         Some-Private-Extension-Tag="abc"/>
</FDT-Instance>
```

Figure 5.1: FDT-Instance example.

Figure 5.1 shows an example FDT-Instance [82]. Two files are described, of type html and mp3 (*Content-Type*), and mapped onto TOIs 1 and 2 respectively. The html file has been content encoded before transmission, i.e. the FLUTE sender has compressed it using "gzip" (*Content-Encoding*). The size of the uncompressed file is "6100" bytes (*Content-Length*), but the actual transport size is smaller. The MD5 checksum (*Content-MD5*) of the uncompressed content allows receivers to check the integrity of the received file. Additional attributes can be defined for application specific purposes, as pointed out with the *Some-Private-Extension-Tag* attribute. The FDT-Instance has a limited validity period (*Expires*), which represents the time in seconds relative to January 1st 1900, 0h (i.e. the 32 most significant bits of a 64 bit NTP time) when it will time out.

5.3 FLUTE in Practice

5.3.1 Delivery Models

FLUTE can be used differently according to packet and file scheduling. In the basic use case, the content is only sent once, each file one after another, as depicted in figure 5.2. The content is static (i.e. does not change during transmission) and the FDT-Instance is delivered before the first file. Each receiver can then choose and download the content it is interested in. This corresponds to a *push* delivery model (section 1.1.2) where all receivers need to be ready before transmission starts. With this model, loss recovery capabilities are limited, since the session may not last long enough for some receivers to entirely recover from losses. Therefore an additional loss recovery mechanism may be required, (e.g. one such file repair mechanism is provided for 3G MBMS, section 6.2).

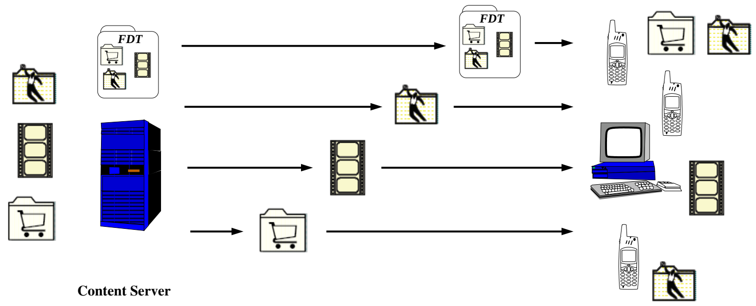


Figure 5.2: One shot delivery (push model).

In *on-demand* delivery the files are typically transmitted in a carousel, cyclically over a longer time period, as depicted in figure 5.3. If the content does not change in this use case, receivers can join and download the content at any time, and all losses will finally be recovered just by listening - providing the session is sufficiently long. In order to make the download performance independent of the loss model, the packets of all files may be transmitted in a random order.

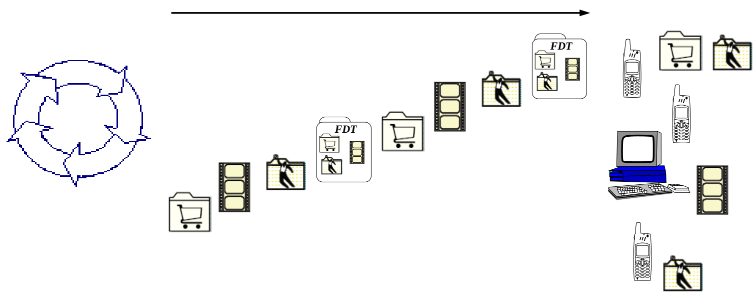


Figure 5.3: Static carousel (on-demand model).

However, if a file changes dynamically, it must be described in a new FDT-Instance and the file must be changed in the carousel. The current cycle can then be stopped in order to make the new file available rapidly, or the sender may choose to wait for the next carousel cycle. This is depicted in figure 5.4.

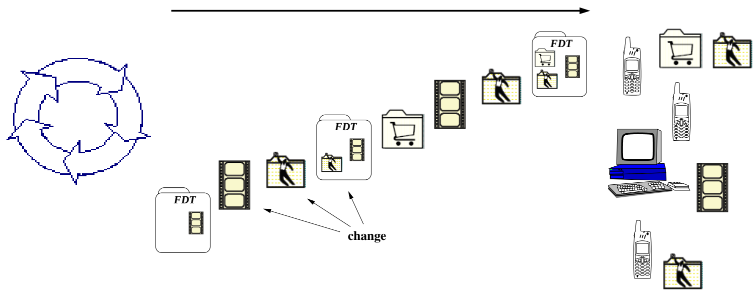


Figure 5.4: Dynamic carousel.

5.3.2 Session Description

To inform potential receivers of an ongoing FLUTE sessions and in order to communicate all FLUTE and ALC session parameters to the receivers, [65] explains how to use the Session Description Protocol (SDP) syntax to describe the parameters required to begin, join, receive data from, and end FLUTE sessions.

5.3.3 Service Announcement

Service announcement is the server initiated delivery of service descriptions (or metadata). In the context of massive user group sizes, often interested in the same mass media content at the same time, the natural solution is to use a multicast solution. The basic need for the delivery of metadata is control of time of reception in relation to the media or service it describes. This is not only limited to the timing of transmission, but also to the reliability of the transmission. Thus, service announcement lends itself very well to the concepts of massive multicast transport of discrete objects with variable error tolerant redundancy needs , and FLUTE addresses exactly these problems. Hence, more recent work relating to "Internet Media Guides" (IMGs) [80] has proposed the use of FLUTE as the primary means to announce metadata over multicast networks.

Chapter 6

Large Scale Content Delivery for Mobile Devices

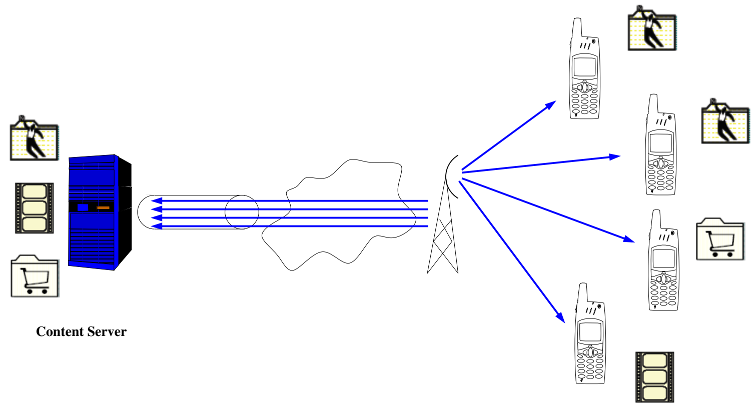
Contents

6.1 An Example: Olympic Games

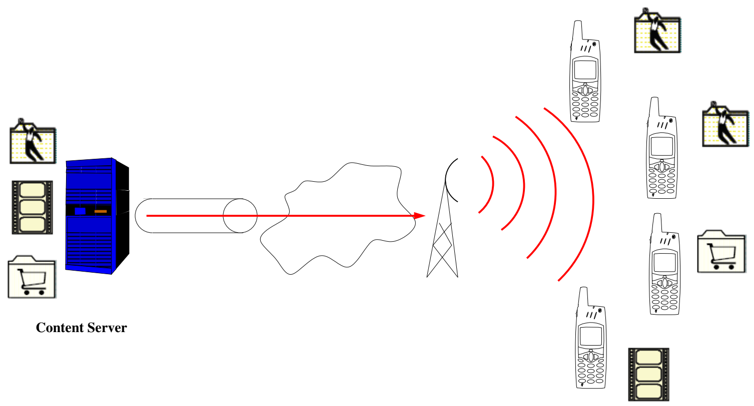
During a big event like the Olympic Games, a lot of digital content is produced and made available to the public. The *content is extremely heterogeneous*: live video and audio streams, sport event results (archived and in real-time), recorded interviews and sport events, timetables of upcoming events, programs/contents guides, etc. Due to the popularity of the Olympic Games, there are potentially *thousands or millions of clients accessing the same content at the same time*. Bringing all this content to the clients is a challenging task, and in this example, we assume that clients are equipped with UMTS/MBMS and/or DVB-H mobile terminals.
UMTS features a bidirectional channel which makes it possible to explicitly request and receive a content over a point-to-point connection. However this is not necessarily the most effective solution. In particular, it raises *scalability issues:* with a high number of clients, (1) the server must support a large number of simultaneous downloads, handle potentially a lot of retransmissions requests (the wireless channel can be a very lossy one), and (2) the current WCDMA wireless channels, where finite bandwidth is shared among all clients, do not yet handle a very high numbers of simultaneous high speed downloads. This is depicted in figure 6.1.
Using a point-to-point approach also raises *power consumption issues:* for a mobile device, using the uplink channel consumes significant power as the device acts as the radio transmitter (other reverse applies for reception), even if only a very small number of feedback messages are sent. Since mobile devices continue to have limited battery life as new mobile applications eat additional power at a comparable rate to the gains battery storage capacity, this issue has to be taken very seriously.
Due to these considerations, a popular content (e.g. the results of the 100m finals), requested simultaneously by a high number of clients, should be broadcast rather unicast to each client independently to counter the problems raised above. Moreover this perfectly matches the broadcast nature of the radio channel.

6.2 MBMS and IP Datacast Services

The Multimedia Broadcast Multicast Service (MBMS) is an enhancement to the current 3G/UMTS cellular systems providing mass media services over IP Multicast and multipoint radio to distributed



(a) Point to point



(b) Point to multi-point

Figure 6.1: Point to point versus point to multi-point.

mobile user groups. It specifies two primary delivery services, download and streaming, upon which user services can be built. [4] standardized the use of FLUTE for both discrete object (or file) download and the announcement of metadata fragments, and likewise, RTP (Real-time Transport Protocol) for multicast delivery of continuous audio/visual streams (figure 6.2). Additionally MBMS defines out-of-band repair mechanisms, that allow recovery of partially received files of a FLUTE session. This is done via explicit requests to the source and by defining a reception report mechanism.

DVB-H is an unidirectional, high bitrate technology, designed for any kind of content broadcasting. It is an excellent solution to broadcast live video or audio streams, such as in the Olympic games example. Additionally, DVB-H can also be used to broadcast popular content, for instance to make sporting results rapidly available to all DVB-H mobile devices. When a bidirectional channel is needed, for instance to access pay-per-view content or to add further reliability services, then UMTS can be used in parallel to DVB-H. The IP Datacast (IPDC) specifications [26] define the download and streaming services for DVB-H, and are based on FLUTE and RTP respectively, just like MBMS.

The potential of multicast to add a low cost delivery channel to an existing massive base of mobile users, specifically for mass media, is very exciting. Mobile mass media, including MBMS and IP Datacast using DVB-H is likely to offer the ultimate showcase to demonstrate the benefit of large scale immediate content distribution protocols in both technical and commercial arenas.

Practical network implementations of MBMS are expected by the end of 2007, and the first functional mobile terminals supporting MBMS are estimated to be available by the end of 2008. For IPDC/DVB-

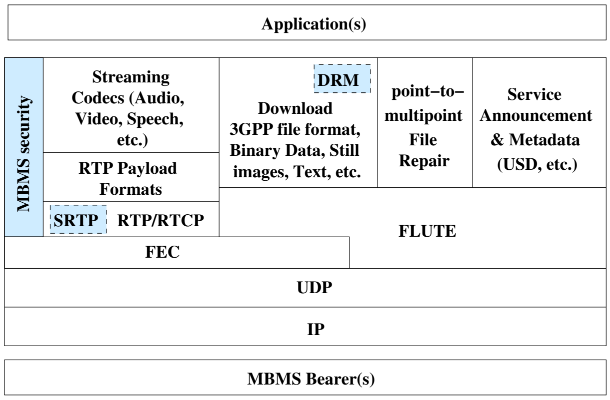


Figure 6.2: The MBMS protocol stack in the downlink.

H trials and pilots already started in many countries from numerous companies. Therefore the first generation of DVB-H services will reach the market much sooner than MBMS.

Part II

Contributions to Large Scale Content Distribution Building Blocks

Chapter 7

Introduction to, Design and Standardization of LDPC Codes

Contents

7.1 Introduction and Motivations

This chapter introduces the large block LDPC/LDGM Codes and its variants used in our work. After a state of the art on LDPC codes we detail LDGM-Triangle, a variant created during this thesis, and present our standardization activities.

7.1.1 Motivations

Why are Reed-Solomon codes not Always Appropriate?

Previous work [15, 58] and our own practical experience have highlighted the importance of FEC and the two limitations of the Reed-Solomon erasure FEC code (or RSE) commonly used and more generally the two limitations of small block FEC codes: (1) the small block size, k, and (2) the high encoding/decoding times. The two parameters (n and k), play an important role in the global efficiency:

- a large n (in fact $n - k$) is favorable because it reduces the probability of symbol duplication at a receiver. For instance sending a new fresh parity symbol to correct different erasures at different receivers is easier if there is a large number of possibilities, i.e. if $n - k$ is large. The extreme case is achieved with the so-called rate-less or expandable codes that can produce an infinite number of parity symbols (i.e. $n \to +\infty$).

- a large k increases the correction capacity of the FEC code. Since a parity symbol can only recover an erasure in the block it belongs to, its erasure capability is therefore inversely proportional to the number of blocks (each of size $\leq k$) a file must be segmented into. Ideally a file may be encoded in a single block, in which case each parity symbol would be useful.

As we have seen in Chapter 2 that RSE is intrinsically limited by the Galois Field size it uses, and therefore the k and n parameters must be limited to small values if high transmission rates are desired. Because of these limitations small block FEC codes are not a recommended solution in most cases, and large block FEC codes or expandable codes should be used instead. Unfortunately, it is not clear at all whether IPR-free expandable codes are feasible or not, since many patents already exist, and we therefore focus on large block FEC codes, which are less impacted by IPRs.

7.1.2 Goal of this Work and Choices Made, in Particular Concerning IPR

In this work we focus on several large block FEC codes derived from LDPC. These codes have two main advantages: (1) they use XOR operations for high speed encoding/decoding, and (2) they operate on large source blocks ($k \gg 1$).

A major issue when working on large block FEC codes are IPR claims, since many patents (in particular US-Patents) have been issued on various aspects. This is for instance the case of the use of irregular graphs [51] in LDPC derived FEC codes, which is covered in particular by [53, 55]. In the present work we deliberately *do not consider techniques which are known to be patented, no matter how efficient they may be*. This choice stems from our desire to *design patent-free, open source FEC codecs*. We want to show that the performance level of these codes are already good enough for them to be used in several environments (and we remain rather confident on the possibility to further improve them).

Unlike many other works in the coding theory area, our work deliberately skips theoretical aspects to focus on practical results (presented in the following chapters). This standpoint is only rarely considered and [25, 88], our preliminary work [94], and to some extent [15], are the only similar works we are aware of.

7.1.3 Standardization Activities

Digital Fountain has put a lot of effort in standardization activities, in order to let their Raptor code be established as a standard. They are very active in 3GPP MBMS where the their Raptor code has become mandatory [4], and are also pushing their code at DVB-H and the IETF [57].

We pushed for the FEC codes presented in this chapter, with special focus on LDGM-Triangle (section 7.3) and LDGM-Staircase (section 7.2.2). We have published an Internet-Draft on this issue, describing a FEC Building Block instantiation for an Under-Specified FEC Scheme for the broad class of LDPC codes [98] and presented the work at the IETF. There is a strong incentive at the RMT working group to have several FEC codes standardized (i.e. offer alternatives to the Raptor code). Our LDPC codes have therefore a strong probability to become one of these standardized codes.

[98] introduces the Under-Specified FEC Encoding ID 132, and defines the FEC OTI and its associated operations for LDPC codes. Encoding, decoding algorithms as well as the parity check creation algorithm for LDGM Staircase and LDGM Triangle are described in [98].

In parallel we proposed the LDGM-Staircase codes at DVB-H. Discussions are in progress.

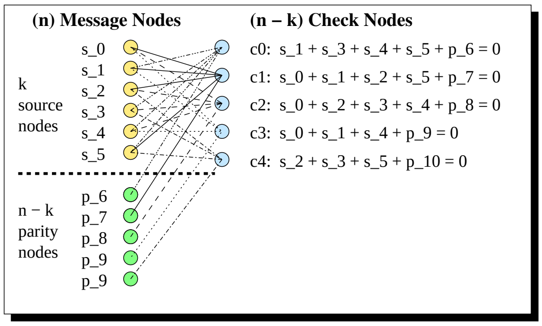
This chapter is structured as follows: section 7.2 gives an introduction and a state of the art of LDPC codes and the derivates used in our work, section 7.3 described LDGM-Triangle, a variant of LDGM/LDPC discovered during the thesis, and finally section 7.4 concludes this chapter.

7.2 The LDPC/LDGM FEC Codes

Low Density Generator Matrix (LDGM) codes are variants of the well known LDPC codes introduced by Gallager in the 1960s [31].

7.2.1 LDGM Codes

The Low Density Generator Matrix (LDGM) code [94] is a linear $(n; k)$ block code. It creates linear equations between the k source symbols: s_i, $i \in \{0..k-1\}$ and the $n-k$ parity symbols: p_i, $i \in \{k..n-1\}$. More precisely each parity symbol is equal to the sum of a subset of the source symbols, as defined in the linear equations. The relationships specified by these equations can be represented using one of two equivalent representations: a *bipartite graph* (A.K.A Tanner Graph) and a *parity check matrix* (figure 7.1).



(a) Bipartite graph

$$
\begin{array}{c}
\quad\quad\quad\quad\quad\quad s_0 \ \cdots \ s_5 \ \ p_6 \ \cdots \ p_{10} \\
[\, H_1 \mid Id_5 \,] = \left(\begin{array}{cccccc|ccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
\end{array}
$$

(b) H matrix

Figure 7.1: A regular bipartite graph and its associated parity check matrix for LDGM.

Bipartite Graph

The bipartite graph defines relationships between left nodes, called message nodes, and right nodes, called check nodes. The k source symbols constitute the first k message nodes, while the parity symbols constitute the remaining $n-k$ message nodes. The check nodes represent relationships between the various symbols (but are not themselves symbols) and form a set of linear equations. For each edge in this graph between a source message node and a check node, the corresponding source node is sum'ed (XOR'ed) in the associated constraint. In the LDGM approach, all the parity message nodes are linked to *exactly one* check node (it is not the case with other codes). Since we only consider regular codes, the degree of each source message node, also called *left degree*, is the same, and the degree of check nodes is on average equal to:

$$right\_deg_{LDGM} = 1 + \frac{k \times left\_deg}{n-k}$$

When using a high FEC expansion ratio (i.e. a large $n-k$ denominator in the above equation), there may be check nodes (i.e. equations) connecting less than two source symbols. In such a case, we automatically add one or two extra edges in the bipartite graph with randomly selected source symbols. Therefore the average right degree is at least 2 in the above equation ($right\_deg_{LDGM} = max(2; ...)$). The upper part of this bipartite graph is built randomly, following an appropriate left and right degree distribution (in our case a left degree of 3 and a right degree of at least 2, not considering parity nodes). The lower part of this graph follows other rules, depending on the LDGM variant. With LDGM, figure 7.1(a), each parity node is linked to one check node with LDGM-Staircase, section 7.2.2, each parity node is linked with two successive check nodes .

Parity Check Matrix

A dual representation consists in building a parity check matrix, H (figure 7.1(b)). With LDGM, this matrix is the concatenation of matrix H_1 and an identity matrix Id_{n-k}. There is a 1 in the $\{i; j\}$ entry of matrix H each time there is an edge between message node j and check node i in the associated bipartite graph.

H forms a system of linear equations between source symbols, s_i, $i \in \{0..k-1\}$ and parity symbols: p_i, $i \in \{k..n-1\}$. The basic operator is `XOR`.

Parity Check Matrix Creation

The H1 sub-matrix is filled using a two step algorithm. First we randomly choose $left\_degree$ "1s" per column ($left\_degree$ defaults to 3). Then, we check, for each line (i.e. each check node, or equation), that there are at least two "1s", and if needed we add one or two "1s" randomly on this line. Indeed, when using a high FEC expansion ratio, some check nodes may connect less than two source packets since the first step of the filling algorithm always adds exactly $left\_degree * k$ "1s" in H1, independently of the number of lines $(n-k)$ of H1.
We do not try to remove degree four cycles in the parity check creation process, even if this is known to be sub-optimal. The main motivations for this choice are (1) we do not want to compromise encoding and decoding performances (removing degree four cycles is a CPU intensive task), and (2) we observed little influence over the overall performances, which is confirmed by [25].
These parity check creation rules also apply to the other LDGM or LDPC codes introduced in the coming sections.

Encoding Algorithm

Thanks to the `XOR` properties and the simple structure of the LDGM matrix, *parity symbol creation is straightforward and extremely fast*: each parity symbol is equal to the sum of all source symbols in the associated equation. For instance symbol p_7 is equal to the sum $s_0 \oplus s_1 \oplus s_2 \oplus s_5$.
The fact that this matrix can be used for encoding, or in other words the fact this matrix is a "Generator Matrix", accounts for the "GM" acronym in the LDGM name. Besides, the H matrix is sparse[1]. This is due to the fixed number of 1s per column, no matter the matrix size, or said differently, no matter the block size (k) and number of parity symbols created ($n-k$). This "low density" property accounts for the "LD" acronym in the LDGM name. Because of these two properties, *encoding is extremely fast with LDGM and its derivatives.*

Decoding with the Iterative Decoding Algorithm

Decoding follows a trivial algorithm. 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. We then replace this variable by its value in all remaining equations and reiterate, recursively. The value of several variables can therefore be found if lucky.
In our case, the set of constraints (check nodes) form a set of k linear equations of n variables (source and parity symbols). As such this system cannot be solved and we need to receive symbols from the network. Each incoming symbol contains the value of the associated variable, so we replace this variable in all linear equations in which it appears. We then apply the above algorithm and see if decoding can progress by one or more steps. As we approach the end of decoding, incoming symbols tend to trigger the decoding of several symbols, until all of the k source symbols have been recovered (but not necessarily all of the $n-k$ parity symbols).
The detailed algorithm and its implementation is described in detail in chapter 8.

7.2.2 LDGM-Staircase Codes

LDGM-Staircase codes (suggested in [61]), is a trivial variant of LDGM [2]. LDGM Staircase only differs from LDGM by the fact that the Id_{n-k} matrix has been replaced by a "staircase matrix" SC_{n-k} of the same size (figure 7.2). An example of an $(400;600)$ matrix is given in figure 7.3.
This small variation does not affect encoding, which remains a simple and highly efficient process: each parity symbol is equal to the sum of all source symbols in the associated equation, plus the previous parity symbol. The only constraint is that encoding must follow the natural parity symbol order. For

[1]Note that this sparse property is not visible in figure 7.1 which has a limited size for practical reasons.
[2]The author of [61] refers this code as LDPC Staircase, we prefer the LDGM Staircase name because of its closeness to LDGM and the fact that the parity check matrix H is also a generator matrix

$$
\begin{array}{c}
\begin{array}{cccccc|ccccc} s_0 & & \dots & & & s_5 & p_6 & & \dots & & p_{10} \end{array} \\
[\, H_1 \,|SC_5] = \left(\begin{array}{cccccc|ccccc}
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1
\end{array}\right)
\end{array}
\quad
\begin{array}{l}
s_1 + s_3 + s_4 + s_5 + p_6 = 0 \\
s_0 + s_1 + s_2 + s_5 + p_6 + p_7 = 0 \\
\quad \dots
\end{array}
$$

Figure 7.2: Parity Check Matrix H of an (6;11) LDPC-Staircase code.

Figure 7.3: Parity check matrix (H) for LDGM-Staircase (k=400, n=600).

instance encoding starts with symbol p_6, and then symbol p_7 equal to the sum $s_0 \oplus s_1 \oplus s_2 \oplus s_5 \oplus p_6$, etc. Yet this simple variation *largely improves the decoding efficiency* as shown in chapter 10.
Intuitively, parity symbols are now protected and an erased parity symbol, for instance symbol p_7, can be recovered thanks to symbol p_8 since they are both linked by a common constraint. On the opposite, with LDGM, an erased parity symbol cannot be recovered, unless all the associated source nodes in the associated constraint are known, but in that case this is useless! Decoding follows the same algorithm as LDGM.
Given the right degree, the left degree of check nodes is on average equal to:

$$right\_deg_{LDGM\_Staircase} = 2 + \frac{k \times left\_deg}{n - k}$$

7.2.3 LDPC Code

LDPC differ from the previous codes by the fact the H_1 matrix encompasses *all* message nodes, source and parity symbols (so H and H_1 are the same). A parity symbol now takes part in several constraints randomly selected.
This is an asset from a decoding point of view since an erased parity symbol may be recovered by several different ways (as with LDGM Staircase). But it has a cost since there is no trivial encoding scheme and the parity check matrix cannot be used directly for producing parity symbols. Encoding requires that the set of $n - k$ linear equations be solved, where the $n - k$ parity symbols take the role of variables. Several schemes exist to that purpose [61] that all produce a generator matrix, G, solution of the system, but even with optimized schemes, it remains a time consuming task. Encoding now consists in multiplying this G matrix with the vector composed of source symbols. Since G has no reason to be sparse (unlike H), this multiplication is yet another time consuming task. Non surprisingly, *encoding is rather slow with LDPC compared to LDGM codes.*

Decoding follows the same algorithm as LDGM and operates on the H matrix.
Because of our choice of not using patented irregular graphs, we only consider in this work regular (3, $right\_deg\_LDPC$) LDPC codes, where the left nodes of the bipartite graph are of degree 3 and right nodes of degree:

$$right\_deg\_LDPC = \frac{n \times left\_deg}{n-k}$$

7.3 LDGM-Triangle Codes

One contribution of this thesis is the invention and design of LDGM-Triangle Codes, a variant of LDGM-Staircase. The empty triangle of LDGM-Staircase beneath the staircase diagonal is now filled, following an appropriate rule:

```
// Let's consider row i.
// Create the staircase first...
add a 1 in row i and column k+i;
if (i > 0)
    add a 1 in row i in column k+i-1;
// Then the triangle... Add at most i-2 "1"s.
tmp_idx = i - 2;
for (n = i - 2; n > 0; n--) {
    tmp_idx = rand() modulo tmp_idx;
    if (tmp_idx > 0)
        add a 1 in row i and column (k+tmp_idx);
    else
        break;
}
```

This rule leads parity nodes with a low index number to have a higher degree than parity nodes with a higher index number, as shown in figure 7.4. This variation increases performances compared to LDGM Staircase for small FEC expansion ratios as will be shown in chapter 10. Intuitively, our algorithm adds a "progressive" dependency between parity symbols, i.e. that only some parity symbols protect a large number of symbols[3].
Here also encoding must follow the natural parity symbol order (this is the reason why only the lower triangle is filled). Otherwise it remains highly efficient, even if a bit slower than with LDGM Staircase since there are more "1"s per row.
Decoding follows the same algorithm as all LDGM codes and is therefore very fast.

7.4 Conclusions

In this chapter we presented the basic principles of LDGM and LDPC codes. It is a class of codes which has very fast decoding and encoding speeds and very respective error correcting capabilities, what will be confirmed in the following chapters.

[3]It is interesting to note that experiments (not included in this work) have shown that filling the triangle in a completely random manner, without progression in the density, leads to lower performances than LDGM Staircase.

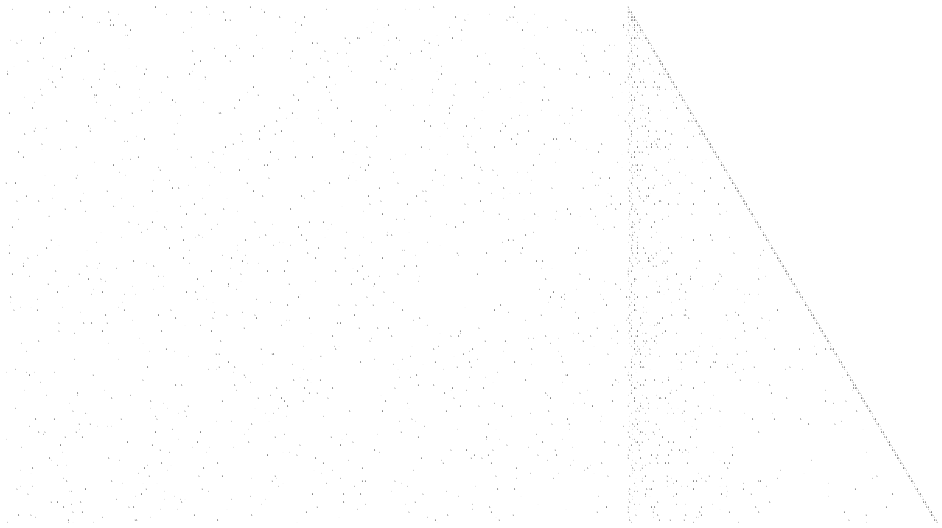


Figure 7.4: Parity check matrix (H) for LDGM-Triangle (k=400, n=600).

Chapter 8

Implementation and Evaluation of a LDPC FEC Codec

Contents

8.1 Introduction

This chapter presents and evaluates a software implementation of a large block FEC codec for the erasure channel. All codes presented in Chapter 7 have been implemented, we however focus on the two codes LDGM-Staircase and LDGM-Triangle.
Many previous work done on large block FEC codes have led to several proprietary, patent protected, codes. So we did our best to identify an IPR-free class of solutions, even if it sometimes required to ignore schemes known to improve the codec efficiency. To the best of our knowledge, our codec does not infringe any IPR and we invite anybody who disagrees to contact us immediately.
During the codec design, we put a lot of efforts on optimizing both the encoding/decoding speeds and the maximum memory requirements. Our results show that encoding and decoding are extremely efficient, even on small devices, and memory requirements are acceptable. These results open its use to a wide field of applications and scenarios, including high speed networks and/or constrained devices (e.g. PDA). We therefore believe that our codec offers an interesting and competitive alternative for many applications requiring an open-source, IPR-free, large block FEC codec.
The first motivation for our LDPC codec implementation is our *MultiCast Library (MCLv3)* [93] (see Chapter 13) which implements the ALC protocol family and the FLUTE protocol/application, and provides an integrated solution for the reliable and highly scalable delivery of content (e.g. files) over

any kind of unidirectional or bidirectional network. Our FEC codec may however be used with any other type of application requiring a fast large block FEC code.

8.1.1 On the Use of Other Large Block FEC Codes

If the present work only considers LDGM-Staircase and LDGM-Triangle codes, this is not restrictive. Other more efficient codes are progressively added to our implementation, as soon as they are identified, implemented, and tested. The present codec must therefore be viewed as a *general purpose but efficient framework for the implementation of various large block FEC codes.* The stress is essentially put on finding good data structures and algorithms for the high performance of these codes (essentially from a maximum memory requirements and encoding/decoding speed points of view), as well as on their practical use, rather than on the intrinsic performances of each code.

8.1.2 Software Implementation versus Hardware Implementation

LDPC codes have been adopted for the DVB-S2 standard, and they are considered in IEEE 802.16 broadband wireless transmission schemes [20]. Conversely to our implementation these codes operate on a binary symmetric channel (BSC) rather than on an packet erasure channel, i.e. that forward error correction is made at bit-level rather than at packet level. Codes operating on a BSC are more likely to be implemented in hardware rather than in software. This is possible since the standards like DVB-S2 and IEEE 802.16 only define a very small number of predefined H matrices, code rates and block sizes, to which the hardware implementation has to comply. In order to support other code rates or block sizes, the codec has to use several H matrices in parallel or to concatenate several codes. The codec efficiency is less interesting in this case.

Implementing LDPC at software level as we did, implies many challenges that mostly come from the fact that the codec needs to be highly flexible. This flexibility can not be supported in a hardware implementation because of its constrained and inflexible resources (i.e. memory). Conversely, software implementation has a larger amount of memory available, which also means that it can operate on large blocks. Therefore, for software implementations the code rate and the block size is an adaptable parameter. Moreover the H matrix is not a predefined one, but is generated for each use case using a seed or by defining new matrix generation algorithms.

Since the H matrix, the code rate and the block size are not predefined, adequate data structures are needed that support this high flexibility. These data structures are described in detail in the next sections.

One major drawback of a software level implementation is that encoding and decoding speed will be below the speeds of the hardware implementation, since no dedicated hardware acceleration is available and massive parallelism, as often used in hardware implementations, is not possible.

8.1.3 Packets and Symbols

As we have already pointed out in chapter 2 a FEC code operates on symbols, which are in most use cases an entire packet, but it is also possible to operate on symbols smaller than one packet size. For simplicity in this chapter we will refer to symbol and packets indifferently, i.e. we assume one symbol is equal to one packet. The codec was originally designed to only support this feature, and the internal variables still uses this terminology. It is however possible to use the present codec in the case when a symbol is smaller than one packet.

This chapter is structured as follows: section 8.2 describes the implementation details, i.e. the data structures and algorithm used; section 8.3 experimentally evaluates the performance of our implementation. Section 8.4 gives a quick glance at the API of the codec and finally section 8.5 concludes this chapter.

8.2 Implementation Details

Our C++ LDGM-Staircase and LDGM-Triangle codec [77] is implemented as a general purpose library. This library, which offers a simple API that will be introduced later on, can then be linked

by an application, like our FLUTE/ALC implementation [93].
Our main focus when designing the codec, was to optimize memory consumption and decoding/encoding times. Therefore the matrix creation and storage scheme, and the decoding algorithm implementation details, are two critical aspects for achieving high performances.

8.2.1 The H Matrix

H Matrix Creation

The codec creates a regular parity check matrix H given the $k, n, left\_degree$ and a *seed*. The seed initializes a Pseudo-Random Number Generator (PRNG) used to fill the parity check matrix.
The H1 sub-matrix is filled using the two step algorithm described in section 7.2.1.
The second sub-matrix, on the right part of H, is then created, using the LDGM variant specific features.

H Matrix Storage

The parity check matrix has to be stored in an efficient way. We used for that the library included in R. Neal's software [67]. The H parity check matrix is stored in a sparse array: to each "1" of the matrix is associated an entry which contains its row and column indexes, plus a pointer to the following "1" on the left, on the right, upward (except with LDGM-Staircase codes where the upward pointer is never used) and downward. This sparse structure has several advantages:

- it enables *major memory savings* compared to a dense structure. For instance a $(n = 30000; k = 20000)$ code leads to a H matrix of size 10000×30000 which requires at least 37.5 MBytes of storage with a dense representation and only $\approx$ 2.5 MBytes using a sparse structure with LDGM-Staircase.

- it enables a *very fast encoding*, since it is sufficient to follow the right pointer of a line to find all the source/parity packets that must be XOR'ed to produce a given parity packet.

- similarly, it enables a *very fast decoding*, since it is sufficient to follow the "down" pointer of a given column (i.e. packet) to find all the equations where it takes part.

Note that with LDPC, the sparse structure is not appropriate for storing the G generator matrix (which has no reason to be sparse), and in that case a dense structure is used. This has a cost which is in parts responsible of the limitations observed during experiments in chapter 10.

8.2.2 The Iterative Decoding Algorithm

Decoding follows the algorithm pointed out in section 7.2.1: 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 (A.K.A. partial sum). We then replace this variable by its value in all remaining equations and reiterate, recursively. The value of several variables can therefore be found if lucky.
In our case, the set of check nodes form a set of $n - k$ linear equations of n variables, the k source and $n - k$ parity packets. As such this system cannot be solved and we need to receive packets from the network. Each incoming packet contains the value of the associated variable, so we replace this variable in the linear equations in which it appears. Replacing a variable means adding (`XORing`) the value of the new packet to the partial sum of the equation. For optimal performances, this addition is performed 32 bit at a time (or 64 bits at a time on 64-bit architectures), for all words of the packet (we require that the packet size be multiple of 32 bits). We then apply the above decoding algorithm and see if decoding can progress by one or more steps. As we approach the end of decoding, incoming packets tend to trigger the decoding of several packets, until all of the k source packets have been recovered. Note that decoding will probably stop before all of the $n - k$ parity packets be received/decoded, since parity packets are only temporary variables, and their decoding not an objective.

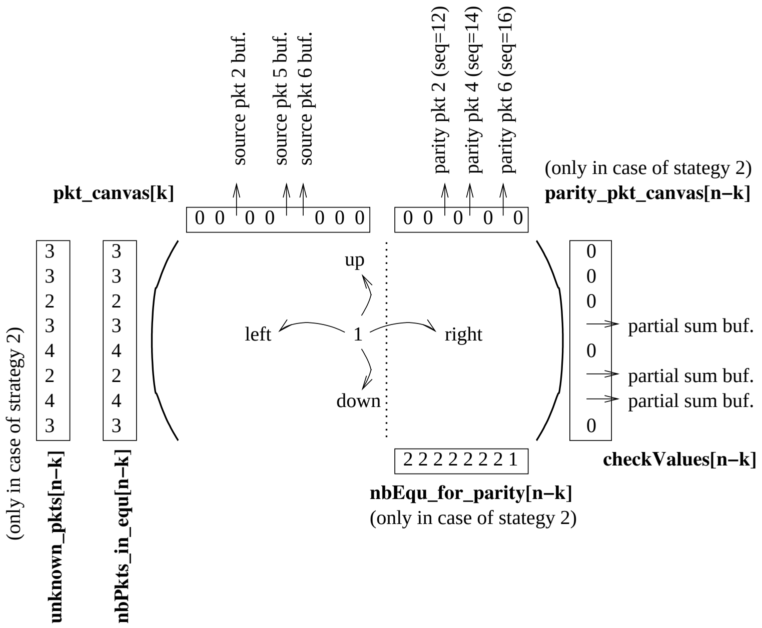


Figure 8.1: Data structures used by the decoding algorithm.

Data structures Used by the Decoding Algorithm

To implement the above algorithm several data structures and tables are needed. In this evaluation the codec assumes that all information (including packets) is stored in physical memory (RAM), which creates a practical limitation for huge objects. This issue is very critical when using LDPC since the memory consumption is even higher. Some experiments in chapter 10 with LDPC could not be carried out because of this (section 10.2.3). Recent work addresses this aspect and the implementation now supports the dedicated virtual memory framework included in MCLv3 [93] (see section 8.2.3). This implementation was however not available at the time of the experiments and therefore this evaluation does not consider this framework.

There is anyway a limitation because the random nature of the H/G matrices requires at any time a fast access to any source packet (and for LDPC and LDGM-Triangle codes also to any parity packet) during the encoding and decoding processes, which is only possible if most of the packets are kept in memory.

The decoder has to store several types of data (Figure 8.1):

- *source packets*: All source packets will be delivered to the application, and therefore all received or decoded source packets are systematically kept in memory. An array of pointers, `pkt_canvas[]`, is introduced to point to the received or decoded source packets. The index in this array is the packet sequence number.

- *parity packets*: On the opposite, parity packets are not delivered to the application, they are only temporary elements in the decoding process. So, at first sight, there is no need to store the received (or decoded) parity packets at the decoder, and their value can be simply summed to the corresponding partial sums.

  However we will see later that an optimization exists that stores parity packets rather than injecting them in the corresponding partial sums. To that purpose, an array of pointers, `parity_pkt_canvas[]`, is introduced to point to the parity packets that must be stored. The index in this array is the parity packet sequence number minus k (said differently, packet k is the first parity packet, and is stored in `parity_pkt_canvas[0]`).

- *partial sums*: Finally partial sums for a given equation (or check node), must be handled by the decoder. Therefore an array of pointers, `checkValues[]`, is introduced to point to the partial sums in use. Yet the partial sum buffers are allocated only when needed, while decoding (the exact criteria depends on the decoding algorithm). In the following this array is referenced as `checkValues[]`, the index corresponding to the packet sequence number.

These different data structures can be manipulated in different ways, according to the implementation of the decoding algorithm. We present two variants and highlight the high practical importance of optimizing the iterative decoding algorithm.

Strategy 1: Adding Systematically Parity Packets to Partial Sums

In this implementation of the decoding algorithm, parity packets are always immediately added to all the concerned partial sums. Therefore there is no need to store parity packets (i.e. the array `parity_pkt_canvas[]` does not exist). Partial sums are allocated only when needed (rather than in advance, at initialization time), as soon as a packet for the associated equation is available. This is the way we implemented the decoding algorithm first, which seemed to us the most natural approach. In the remaining, we will sometimes denote this strategy by "classical approach".
The decoder is composed of only one function, `DecodeFecStep()`, that is called every time a new packet is received or decoded (called recursively in the latter case):

```
ldpc_error_status
LDPCFecSession::DecodeFecStep
   (void*   pkt_canvas[], // source pkt canvas
    void*   new_pkt,      // the new packet
    int     new_pkt_seqno,// assoc. sequence nb
    bool    store_packet, // true if needs to
                          // alloc memory, copy
                          // the packet in it,
                          // & call any callback
    void*   context_4_callback) // opaque context
                          // passed to callback
```

This function performs the following tasks:

1. If `new_pkt` is an already decoded or received packet, the function returns immediately and does nothing.

2. If `new_pkt` is a data packet, it is permanently stored in `pkt_canvas[]`.

3. For each equation i having a degree greater than one (i.e. more than one unknown variable), with an entry in column `new_pkt_seqno` (i.e. having `new_pkt` as a variable), do the following:

   – If `checkValues[i]` is null, allocate a buffer for this partial sum;

   – add `new_pkt` to the partial sum `checkValues[i]`;

   – remove the entry $(i; new\_pkt\_seqno)$ of the H matrix.

   – If the new degree of equation i is one, we have decoded a new packet and have to remember the index of the equation in a list of indexes for newly decoded packets for step 4.

4. For all newly generated packets in step 3, move (i.e. copy and free) `checkValues[i]` into a static buffer[1], remove the last entry in equation i, and call `DecodeFecStep` recursively with the static buffer and the newly generated packet index as parameters.

This algorithm needs to know the degree of each equation (i.e. the number of 1s). For higher performances, we added a `nbPkts_in_equ[]` array, indicating for each equation i the number of remaining variables not XOR'ed in the partial sum `checkValues[i]` (or in other words the remaining number of "1s" for row i of matrix H). This array is filled at initialization time, and each entry is decremented each time a packet is added to the corresponding partial sum.

[1]Using a static buffer is viable here, even if it there is a recursive call, because it will be used before being possibly overwritten in the inner calls.

Strategy 2 (Optimized): Keeping Parity Packets in Memory

The idea of the second implementation variant of the decoding algorithm is to store parity packets, using the `parity_pkt_canvas[]` array, rather than automatically allocating the associated partial sums and adding the parity packet to them. Now, a partial sum is only allocated when needed, when enough packets are available to decode the associated equation. As will be shown in the evaluation, this slight change sometimes largely improves performances. No change is made concerning source packets, here also they are always stored in the `pkt_canvas[]` array.
As with strategy 1, the decoder consists of only one function, `DecodeFecStep`, called every time a new packet is received or decoded (called recursively in the latter case). The operations performed are a little bit different, though:

1. If `new_pkt` is an already decoded or received packet, the function returns immediately and does nothing.
2. If `new_pkt` is a data packet, it is permanently stored in `pkt_canvas[]`. If `new_pkt` is a parity packet, it is stored in the `parity_pkt_canvas[]` array if this packet is involved at least in one equation and if this equation cannot be solved immediately.
3. For each equation i having a degree of two and an entry in column `new_pkt_seqno` (i.e. if enough packets, including `new_pkt`, are available to decode equation i), do the following:
   - If `checkValues[i]` is null, allocate a buffer for this partial sum;
   - for all packets j concerned by equation i, add packet j to the partial sum `checkValues[i]` and remove this entry in matrix H; If packet j is a parity packet and if it is no longer needed by any other equation, free packet j.
   - Remember the index of this equation in a list of indexes for newly decoded packets for step 4.
4. For all newly generated packets in step 3, move (i.e. copy and free) `checkValues[i]` into a static buffer[2], remove the last entry in equation i, and call `DecodeFecStep` recursively with the static buffer and the newly generated packet index as parameters.

As before we use the array `nbPkts_in_equ[]` in this algorithm. However we need additional data structures that contain: (1) the number of unknown packets per equation, and (2) the degree of each parity packet (i.e. the number of entries in the column of matrix H for this parity packet). As before, in order to avoid having to go through the H matrix and counting the entries each time the information is needed, we therefore use the arrays `unknown_pkts[n-k]` and `nbEqu_for_parity[]`, that are initialized at the beginning of the session and updated during decoding.

The Case of LDGM-Triangle With FEC Expansion Ratios ≤ 2.0

During our experiments we noticed that the optimized algorithm performed quite good in all cases except with LDGM-Triangle with FEC expansion ratios ≤ 2.0. Therefore we added a small modification in our algorithm. If we use LDGM-Triangle with FEC expansion ratios ≤ 2.0, we never store the parity check packets in the permanent array (in step 2 of the algorithm). However in step 3 we allocate (if not already allocated) systematically each partial sum concerned by the parity check packet and inject its value in it.

8.2.3 Enabling an external Memory Management

As mentioned before, in this evaluation the codec assumes that all information is stored in physical memory (RAM), which creates a practical limitation for huge objects. However, recent work addresses this aspect and the implementation now supports external memory management, i.e. that all symbols memory allocation/deallocation can be managed outside of the codec. This highly increases the flexibility of the codec, since each application can define by itself where and how symbols are stored.

[2]Using a static buffer is viable here, even if it there is a recursive call, because it will be used before being possibly overwritten in the inner calls.

For instance, the application may define a memory management scheme that stores data on disk, enhanced with a cache system to enable quicker access to data.
The external memory management is enabled by callbacks, which the application of the codec has to instantiate appropriately. More precisely:

- The `AllocTmpBuffer` callback is called each time a temporary buffer is required by the system, e.g. to store a partial sum. This function returns a symbol pointer, and accessing the data buffer requires a call to the `GetData` callback.
- The `GetData` callback is called each time the data associated to a symbol must be read.
- The `StoreData` callback is called each time a symbol's buffer has been updated and must be stored reliably by the memory management system.
- The `FreeSymbol` callback is called each time a symbol (or temporary buffer) is no longer required and can be free'd by the memory management system.

8.3 Performance Evaluation

In this section we evaluate the performance of our codec. We also compare the performances of the two decoding algorithms presented before. The following parameters are considered during tests:

- the *FEC expansion ratio*, fec_ratio, or n/k ratio: the higher the ratio, the more parity packets are created. This ratio is the inverse of the "code rate", defined as k/n and which is widely used in the coding community.
- the *object size*, expressed in 1024 byte packets (a common size in the Internet), rather than in bytes. Since the object is usually encoded as a single block, this size in packets is equivalent to the k parameter of the code.

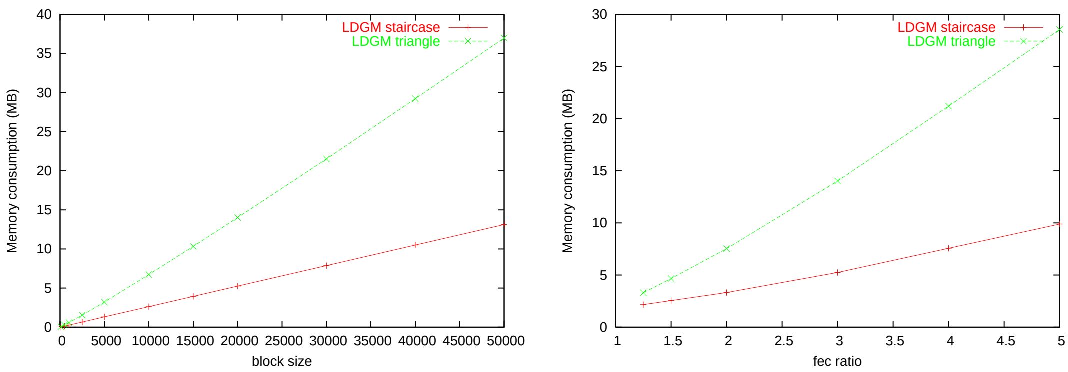
We measure the following performance metrics:

- *encoding/decoding speeds:* measuring the total encoding (resp. decoding) time gives an idea of the maximum transmission (resp. reception) rate from the FEC codec viewpoint.
- *maximum memory requirements:* we measure the maximum memory requirements during encoding and decoding, caused by the need to store source (and often parity) packets, but also all the codec's internal buffers or tables, and in particular the H matrix.

Another measure the *global decoding inefficiency ratio, inef_ratio* is of importance. Since LDGM codes are not MDS, $inef\_ratio * k$ packets must be received for the decoding a block of size k to complete. This ratio, experimentally evaluated, is a key performance metric. Yet, as already explained in section 8.1.1, the stress in this chapter is more on finding good data structures and algorithms rather than on the intrinsic performances of each code itself. So the evaluation of the decoding inefficiency ratios of LDGM-* codes is skipped here and studied in detail in chapter 10.
Note that chapter 10 also includes a comparison study of LDGM-* codes with LDPC and RSE on memory requirements and encoding/decoding speeds.

8.3.1 Testing Application

We designed a basic application on top of these two codecs, derived from the `perf_tool` of our FEC codec distribution [77]. Given an object and a fec_ratio, the application first performs FEC encoding: to k source packets of a block, the codec produces $n - k = (fec\_ratio - 1) * k$ parity packets. Both source and parity packets are 1024 byte long. Once encoding has been performed, the source and parity packets are transmitted in a fully random order. The receiving application receives them one at a time, giving them to the FEC decoder, and stops when the block is fully decoded.
No transmissions on the network are performed, and the same application performs both encoding and decoding. This feature enables us to better control the way packets are "sent", simplifies tests, reduces the overall test durations and enables accurate encoding/decoding time measurements. Note



(a) As a function of the object size, fixed FEC expansion ratio = 3.0

(b) As a function of the FEC expansion ratio, fixed object size = 20000 packets

Figure 8.2: Memory consumption for the H matrix, for LDGM-Staircase and Triangle.

in particular that sending source and parity packets in a random order saves the need for defining a channel loss model (more details on the impact of packet scheduling and the loss model is given in chapter 9). Our results are therefore independent of any particular model, in particular the loss ratio and loss pattern. This is a great asset since the codec may be used in a large number of applications and environments, each of them having specific transmission assumptions. Our results therefore provide average results, rather than specific results.

8.3.2 Memory consumption

We now analyze the *maximum* memory requirements for the different codecs, from the decoder applications points of view. We further distinguish:

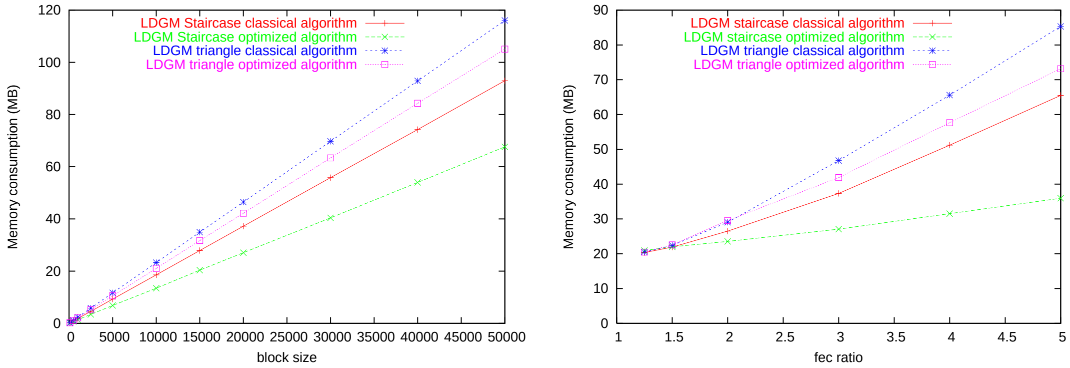
- the memory storage of the H matrix;
- all the remaining memory requirements, essentially for the source and parity packets, the partial sums used by a decoder, and also the various buffers or tables required by the codec.

Memory storage of H

The memory consumption for the H matrices is shown in Figure 8.2. With an object size of $20,000$ packets and a FEC expansion ratio of 1.5, LDGM-Staircase (resp. Triangle) requires 2.5 MB (resp. 4.6 MB). The requirements are higher when the FEC expansion ratio increases, especially with LDGM-Triangle which requires 28.5 MB with a FEC expansion ratio of 5.0, whereas an LDGM-Staircase "only" requires 9.9 MB in that case. We consider that the memory requirements for storing the H matrix, which largely reflects the density of the matrix, is not a major issue with LDGM codes for small to medium FEC expansion ratios, and remains low in front of the memory needed to store packets during decoding. With higher FEC expansion ratios (especially with LDGM-Triangle), this is less true.

Other Memory Requirements

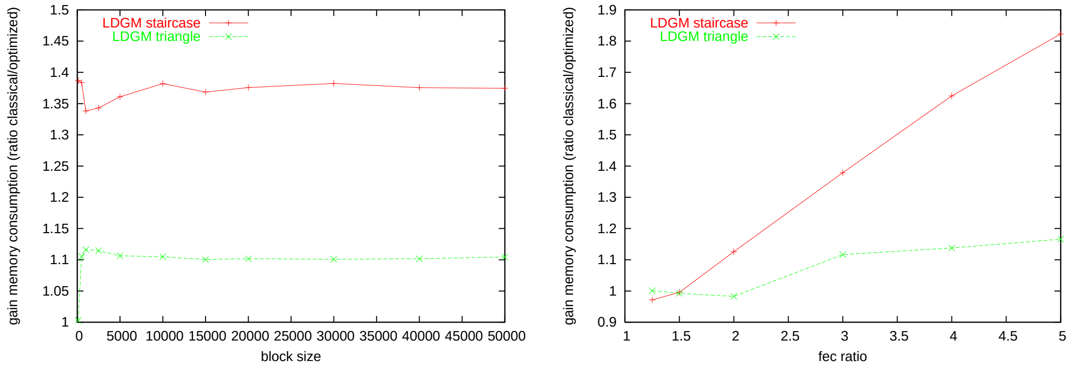
The memory consumption of the decoder (without matrix H) as a function of the FEC expansion ratio and a fixed object size of 20000 packets is shown in Figure 8.3(b). In this figure, we distinguish the two decoding strategies introduced before. We see that the (optimized) strategy 2 provides significant memory savings. With a FEC expansion of 5.0 LDGM-Staircase needs $\approx$ 65 MB with strategy 1 and only $\approx$ 37 MB with strategy 2. With LDGM-Triangle the gains are less important: $\approx$ 85 MB with strategy 1 and $\approx$ 72 MB with strategy 2 in the same conditions.



(a) As a function of the object size, fixed FEC expansion ratio = 3.0

(b) As a function of the FEC expansion ratio, fixed object size = 20000 packets

Figure 8.3: Maximum memory consumption at the decoder (without H), for LDGM-Staircase and Triangle.



(a) As a function of the object size, fixed FEC expansion ratio = 3.0

(b) As a function of the FEC expansion ratio, fixed object size = 20000 packets

Figure 8.4: Memory gain ratio at the decoder, by using strategy 2 rather than strategy 1.

In order to better highlight the gains made possible by strategy 2, Figure 8.4 shows the gain as the ratio $\frac{memory\_used\_by\_strategy\_1}{memory\_used\_by\_optimized\_strategy\_2}$. The gain increases with the FEC expansion ratio achieving a ratio of 1.82 (resp. 1.18) at a FEC expansion ratio of 5.0 with LDGM-Staircase (resp. Triangle).
The memory consumption of the decoder as a function of the object size and a fixed FEC expansion ratio of 3.0 is shown in figure 8.3(a). We notice again that memory savings for LDGM-Staircase are more important. However we also notice in Figure 8.4(a) that the gain is independent of the object size, i.e. the gain ratio is almost constant for a given FEC expansion ratio.

8.3.3 Encoding/Decoding speeds

Impacts of the Decoding Algorithm Strategy

If the main goal of the decoding algorithm strategy 2 was to reduce memory requirements, we found it also provided some benefits with respect to the decoding speed. This is visible in Figure 8.5 which shows the decoding times as a function of the FEC expansion ratio, on a powerful Pentium IV 3.06 GHz, 2GB/Linux host. The reason is that packets are only XOR'ed to the partial sum when needed (i.e. when an equation can be solved) rather than in advance (i.e. when a new packet is available).

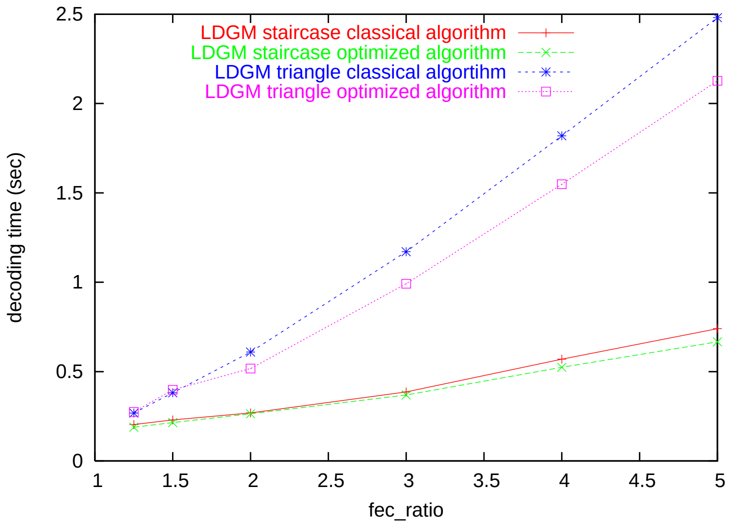


Figure 8.5: Decoding time as a function of FEC expansion ratio, object size = 20000 packets, P-IV/3.06 GHz.

The overall number of sums is therefore reduced to the minimum required. Therefore, in the remaining of this work, we only consider the optimized decoding algorithm.

Encoding/Decoding Speeds on various Architectures

Measuring the FEC codec initialization time and the total encoding (resp. decoding) time enables to measure the maximum sustainable bandwidth. More precisely, two encoding bandwidths can be considered: a global encoding bandwidth, that takes into account both the produced parity packets and the source packets, and measured as:

$$\frac{n * pkt\ size\ (in\ bits)}{time}$$

and a partial encoding bandwidth that only takes into account the produced parity packets:

$$\frac{(n-k) * pkt\ size\ (in\ bits)}{time}$$

The first measure indicates whether the codec can support a given transmission rate when using a systematic FEC code (both source and parity packets are sent), whereas the second one refers to what is actually *produced* by the FEC codec. No such distinction exists at the receiver, and the bandwidth is always measured as:

$$\frac{number\ of\ pkts\ required\, for\ decoding * pkt\ size\ (in\ bits)}{time}$$

Since this value depends on the global decoding inefficiency ratio which varies slightly at each test, we only consider average values.
The encoding and decoding speeds are shown in tables 8.1 and 8.2 respectively. We can notice that encoding/decoding speeds are very high and remains very acceptable even on quite slow machines.

8.4 A Quick Glance at the Codec API

The API is defined by the `LDPCFecSession` class, which offers several methods as shown in figure 8.6. The `InitSession` method is used to specify the main parameters of the codec, in particular the k, n, *seed*, packet size, left degree for source packets, kind of code, and if it is a coding or decoding session. The `BuildFecPacket` method is used to build a (single) new parity packet. With the codes considered in this work, parity packets must be produced *sequentially*, starting at packet k, and up at maximum

Machine	*Code*	*Encoding (parity only)*	*Encoding (source + parity)*
Pentium IV 3.06 GHz 2GB, Linux	Staircase	734.998 Mbps	2,204.990 Mbps
	Triangle	443.813 Mbps	1,331.440 Mbps
Pentium III 1GHz 512 MB, Linux	Staircase	177.954 Mbps	533.864 Mbps
	Triangle	104.823 Mbps	314.470 Mbps
Sun Ultra10, UltraSPARC 480MHz 256MB, Solaris 2.8	Staircase	50.948 Mbps	152.845 Mbps
	Triangle	26.311 Mbps	78.933 Mbps
Compaq IPAQ H3970 PXA250 400Mhz 64MB, Linux	Staircase	22.755 Mbps	68.265 Mbps
	Triangle	12.907 Mbps	38.723 Mbps

Table 8.1: Encoding speeds, FEC expansion ratio = 1.5, object size = 20000 packets.

Machine	*Code*	*Decoding*
Pentium IV 3.06 GHz 2GB, Linux	Staircase	816.505 Mbps
	Triangle	434.866 Mbps
Pentium III 1GHz 512 MB, Linux	Staircase	246.425 Mbps
	Triangle	124.909 Mbps
Sun Ultra10, UltraSPARC 480MHz 256MB, Solaris 2.8	Staircase	56.979 Mbps
	Triangle	34.590 Mbps
Compaq IPAQ H3970 PXA250 400Mhz 64MB, Linux	Staircase	34.256 Mbps
	Triangle	18.952 Mbps

Table 8.2: Decoding speeds, FEC expansion ratio = 1.5, object size = 20000 packets.

```
// creation/initialization/destruction
LDPCFecSession ();
~LDPCFecSession ();
ldpc_error_status InitSession ();
void EndSession ();

// useful methods
void SetVerbosity ();
void MoreAbout ();

// encoding method
ldpc_error_status BuildFecPacket ();

// decoding-related methods
ldpc_error_status DecodeFecStep ();
ldpc_error_status SetCallbackFunctions ();
bool PacketAlreadyKnown ();
bool IsDecodingComplete ();
```

Figure 8.6: The LDPC codec API

packet n. Besides, since the n parameter is specified once and for all in the `InitSession`, there is no way to increase the number of parity packets that may be produced once the codec has been initialized. This is a fundamental limitation of the LDGM codes considered in this work.
The `DecodeFecStep` method, as explained in sections 8.2.2 and 8.2.2, is called each time a new packet is received. In case of a parity packet, upon return of this function, the buffer containing the packet should be free'ed by the caller, no matter what decoding algorithm variant is used (e.g. in case the optimized, strategy 2 is used, a copy of the parity packet may be kept internally in the codec, but this will not be visible by the caller).
Unlike Reed-Solomon codes, there is no way, just by looking at the number of packets received, to decide in advance if decoding is over or not, and this requires counting the number of source packets received and decoded which is a little bit more complex. This is the goal of the `IsDecodingComplete` method.
The `SetCallbackFunctions` is used to register call back functions. Currently, a single callback is defined, `DecodedPkt`. This one is called each time a source packet is decoded after calling the `DecodeFecStep` method. The motivation for this callback is to enable the caller to be informed of new source packets being decoded, and registering them in the caller's specific environment (e.g. in our FLUTE/ALC implementation, we allocate and initialize a dedicate "data unit" (`du_t`) structure for each of them). This callback is given the context pointer that is specified upon calling the `DecodeFecStep` method.
As can be seen, the API has been deliberately kept simple, in order to facilitate its use by applications.

8.5 Conclusions

In this chapter we described the design and implementation of a large block FEC codec, that currently implements the LDGM-Staircase and LDGM-Triangle codes. This codec is easily extensible and new FEC codes are regularly added, once identified and evaluated.
We explained in details the internal design of the codec, and how the implementation of some key functions, like the iterative decoding algorithm, can largely impact the performance achieved.
The codec evaluation focused essentially on the maximum memory requirements, a major concern when working with large blocks, and the coding/decoding speed on various architectures, ranging from PDAs to powerful workstations. We saw that the codec achieved excellent performance from these two points of views with the two LDGM variants currently in use.
Yet this chapter did not include an extensive FEC code evaluation, in particular the decoding inefficiency ratios experienced, since we wanted to focus on the framework more than on the FEC codes themselves. A detailed FEC code error correcting performance evaluation is done in chapter 10.
The C++ codec is distributed under an open-source GNU/LGPL license [77]. It also relies on R. Neal's software [67], which is distributed under an open-source license that grants a free use of the software. Throughout this work, we deliberately avoided techniques known to be patented, no matter how efficient they may be. To the best of our knowledge, our codec does not infringe any IPR and we invite anybody who disagrees to contact us immediately. We believe that it offers an interesting and competitive alternative for many applications requiring an open-source, IPR-free, large block FEC codec.
Even if this codec is already quite useful, and used in several operational environments, we are continuously improving it. The advent of rate-less codes (e.g. Online code [62]) is one possible future direction, since these codes are well suited to many of the target applications of our codec. Yet it is not clear at all whether IPR-free rate-less codes are feasible or not since many patents already exist in that domain.

Chapter 9

Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances: Observations and Recommendations

Contents

9.1 Introduction

The performance of FEC codes largely vary, depending in particular on the code used and on the object size, and these parameters have already been studied in detail by the community. However the FEC performances are also largely dependent on the packet scheduling used during transmission and on the loss pattern introduced by the channel. Little attention has been devoted to these aspects so far.

For instance, sending all source packets first and then parity packets does not necessarily yield the same efficiency as sending the packets in random order. The packet loss distribution observed by a receiver also largely impacts the decoding performances and a given transmission scheme may yield good results for a specific loss distribution and yield catastrophic results in other circumstances. This work analyzes the impact of packet scheduling and loss behaviors on the performances of three FEC codes: Reed-Solomon, LDGM Staircase and LDGM Triangle (the two latter ones based on the implementation presented in the previous chapter). Thanks to this analysis, we define several recommendations on how to best use these codes, which turns out to be of utmost practical importance. For instance it enables to optimize the FLUTE session to a specific download environment, or on the opposite, to find transmission schemes that will behave correctly (but may be not optimally) in a wide set of different environments.
We only consider file delivery applications here, and the transmission latency has therefore little importance. This would not be true with streaming applications. We will not consider the potential impacts of FEC codes and transmission scheme on the decoding latency at a receiver in this chapter. The work of the present chapter is also presented in [75, 76].
The remainder of the chapter is organized as follows: we first introduce the three FEC codes; section 9.2 explains and motivates the modeling method we used; section 9.3 presents and analyzes the performance of several transmission schemes while section 9.4 does the same with a reception model; finally section 9.5 explains how to use these results in practice, then we conclude this chapter.

9.2 Modeling of the Whole System

9.2.1 The Three Models

Let's imagine that a server wants to broadcast a big file using a large scale content delivery system using FLUTE. The content is first FEC encoded which produces additional parity packets. The sender must now decide in which order source and parity packets will be sent, and this choice will largely impact the whole system performances as we will see later. This packet scheduling constitutes the transmission model.
The channel is characterized by a packet loss distribution. It may be a lossy channel with long bursts of packet erasures, or it may be a channel where losses are completely independent from one another. This packet loss distribution constitutes the loss model.

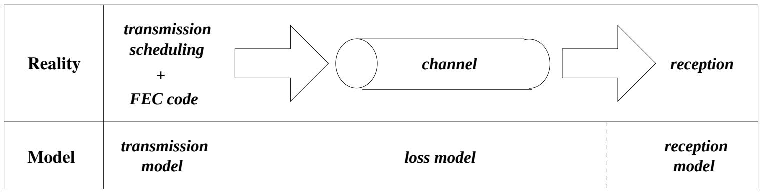


Figure 9.1: Modeling methods.

These two transmission and loss models together characterize a reception behavior at a receiver (Figure 9.1). In case of broadcast applications, there is no reason that different receivers experience the same loss model, so the reception behavior is receiver dependent. This is the approach that will be used for most experimental evaluations (section 9.3).
The reception behavior may also be modeled the other way round, by providing a reception model. This model defines which packets are received (and when) by a receiver. This approach can be complementary to the use of the transmission and loss models, for instance to study the FEC code performances in controlled situations. This is the approach that will be used in section 9.4.

9.2.2 The Channel Model

Finding an appropriate error model for a given channel is a complex task, especially with wireless networks where it is difficult to take into account all parameters (e.g. channel fading, reflections, refractions, diffractions, Doppler effects). Yet in this work we are only interested in a packet loss model (rather than a bit error model) that provides a high level abstraction of the channel parameters. The well known two state Markov model (A.K.A. Gilbert model) is such a simplified loss model, and it is widely used in the literature [8, 107].

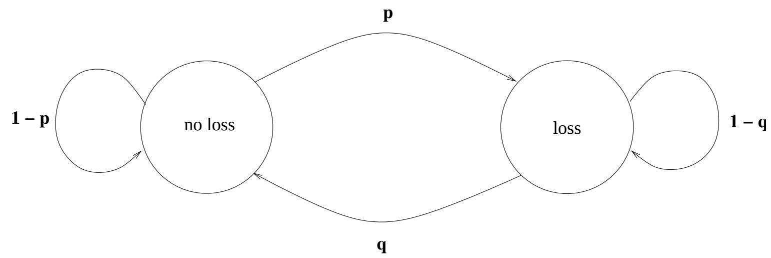


Figure 9.2: Two state Markov loss model.

The model is composed of two states: the *no-loss* state where no packet loss occurs, and the *loss* state where packets are lost (figure 9.2). p indicates the probability to go from no-loss state to loss state, and q from loss state to no-loss state. Having p and q we can calculate the global packet error probability [8]:

$$p_{global} = \frac{p}{p+q}$$

This probability is represented as a 3D-graph in figure 9.3.[1]

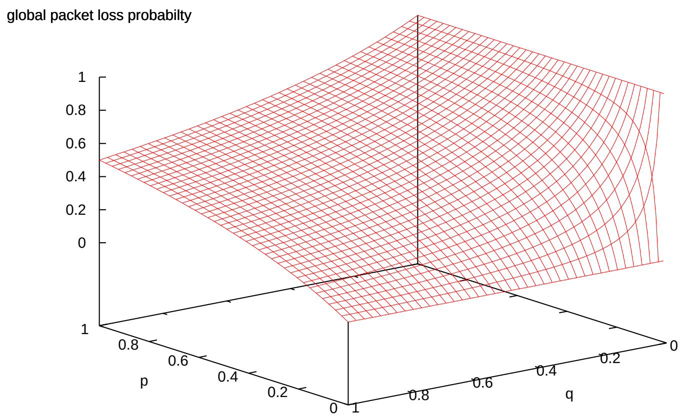


Figure 9.3: Global loss probability.

Depending on the channel modeled, the p and q values will differ. For a given channel it may be possible to determine p and q using packet loss traces. For instance, this has been done in [35] for traces coming from an GSM channel, and in [107] for traces coming from end-to-end connections in the Internet.
The Gilbert model also covers some specific loss behaviors that we want to emphasize:

- No loss: This perfect channel corresponds to $p = 0$.
- Independent and Identically Distributed (IID) losses (A.K.A. the Bernoulli model): This memoryless channel corresponds to $q = 1 - p$.

[1] This graph is helpful in order to have an idea on the p_{global} for the graphs in section 9.3. Note that for better visibility this figure is rotated compared to the figures in section 9.3.

We are aware that the Gilbert model has some shortcomings in error modeling accuracy [35, 107]. We believe that it is however sufficient for our work since it already covers a very large set of loss behaviors. Moreover we took care to perform experiments with a very large set of p and q values (14×14 grid) in order to cover as many channel behaviors as possible. Other more complex models (e.g. the n-state Markov models), that may be required for specific channels, will be considered in future works.

When is Decoding Impossible?

In our work we want to know, given a certain FEC code, how many losses a receiver can support. With the Gilbert model we can calculate the maximum number of packet losses supported by any FEC code. Let's consider a FEC code producing $n-k$ parity packets from k source packets. We then transmit $n_{sent} \leq n$ packets over the network (sending all packets is not mandatory, see section 9.4). The number of packets actually received is equal to:

$$n_{received} = n_{sent} * (1 - p_{global}) \tag{9.1}$$

$n_{received}$ must be at least equal to $inef\_ratio * k$ (remember that LDGM codes are not MDS) for decoding to be successful. When $n_{received} = inef\_ratio * k$ we have:

$$q = \frac{-p * inef\_ratio}{inef\_ratio - \frac{n_{sent}}{k}}$$

These limits are shown in figure 9.4 for FEC expansion ratios 1.5 and 2.5, and assuming $inef\_ratio = 1$ (which is a lower bound for $inef\_ratio$). This figure shows that several areas of the (p, q) parameter space cannot enable a receiver to decode the object. This is not a flaw of the FEC code, it's merely a fundamental limitation.

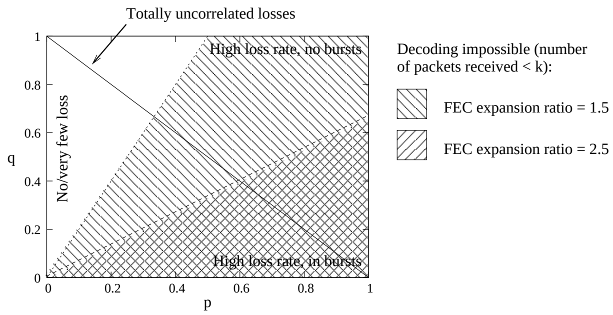
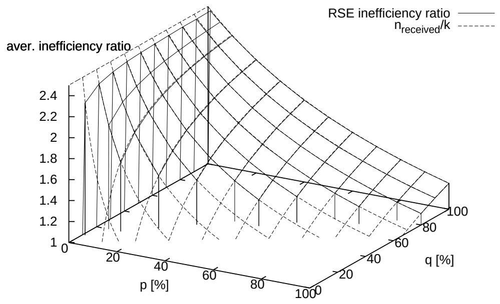


Figure 9.4: Loss limits.

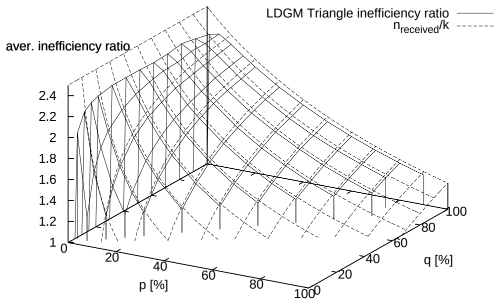
9.3 Simulation Results With Six Transmission Models

9.3.1 Methodology

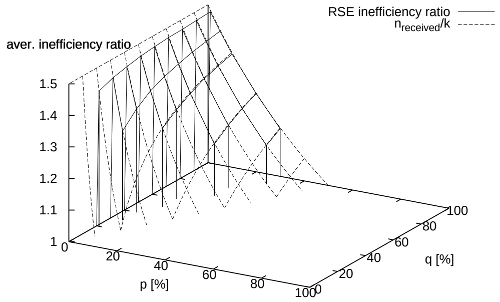
The methodology used to conduct performance tests is the following. We considered the RSE, LDGM Staircase and LDGM Triangle codes, and two typical FEC expansion ratios: 1.5 and 2.5 (they correspond to code rates 2/3 and 2/5 respectively). The object is composed of 20000 packets (i.e. $k = 20000$ with LDGM-* codes).
For RSE we use a Galois Field GF(2^8), i.e. $n \leq 256$. We use the maximum value for n (= 256) and derive the maximum block size supported. The object is then segmented into several blocks using the blocking algorithm described in [104] (see chapter 3).
The performance metric used is the average inefficiency ratio: $inef\_ratio = \frac{n_{necessary\_for\_decoding}}{k}$, which is the total number of packets received when decoding completes, divided by the number of



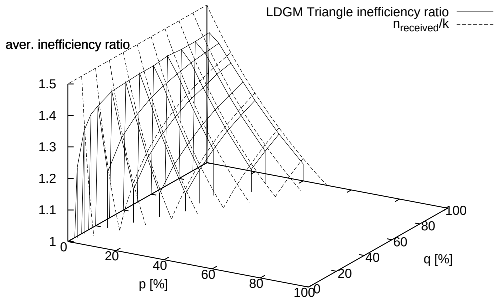
(a) RSE and a FEC expansion ratio of 2.5.



(b) LDGM Triangle and a FEC expansion ratio of 2.5.



(c) RSE and a FEC expansion ratio of 1.5.



(d) LDGM Triangle and a FEC expansion ratio of 1.5.

Figure 9.5: Tx_model_1

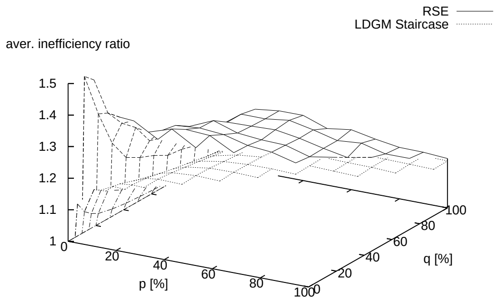
source packets. Of course the optimal value is 1.0, and the higher this *inef_ratio*, the more packets are needed in excess to k for decoding, which is not good[2].

We considered six different *transmission models*. Although this is not an exhaustive study, we believe that these transmission schemes already cover a large set of possibilities and give good insights on the overall performances.

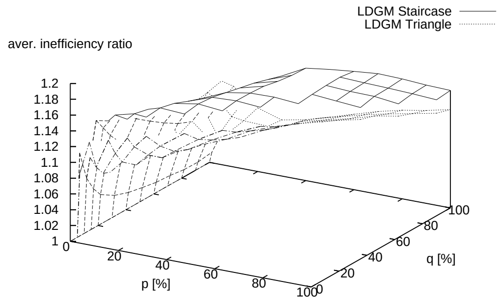
For each transmission model and FEC code, we study the impacts of the channel by varying the p and q probabilities in $[0;1]$ (14×14 values are considered for $(p;q)$). Each point in the resulting 3-D graphs is the average *inef_ratio* value over 100 simulations. Yet if decoding fails for any of the 100 tests, we do not plot any point. This strict strategy enables us to better highlight the areas where the decoding probability is not acceptable. The numerical results of our simulation for the most interesting transmission schemes and codes are reported in [76].

For comparison purposes, we sometimes plot an additional curve, $\frac{n_{received}}{k}$, which corresponds to the total number of packets received (even after decoding stopped) divided by the number of source packets. This is the maximum value that the inefficiency ratio can achieve. It also gives an idea on the number of packets that a receiver may still receive after decoding $(n_{received} - n_{necessary\_for\_decoding})$. When $\frac{n_{received}}{k} = 1$ the curve corresponds to the limits described in section 9.2.2 and figure 9.4. Any additional packet loss will necessarily lead decoding to fail.

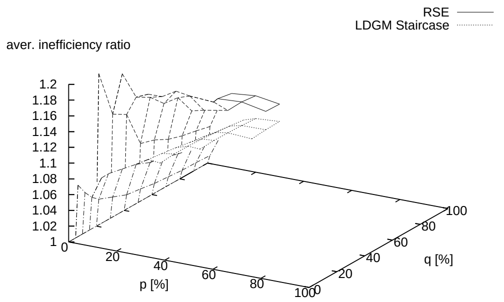
[2] With large block LDGM codes, an inefficiency ratio greater than 1.0 is caused by the non MDS nature of these codes, whereas with RSE, which is an MDS code, this is caused by the coupon collector problem. More information can be found in [97] and in chapter 10.



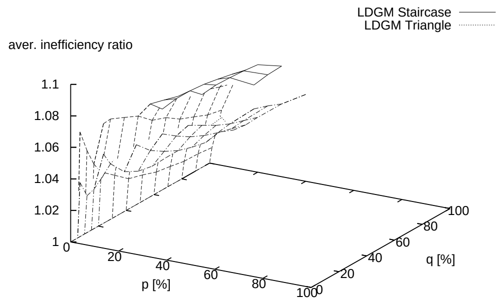
(a) RSE versus LDGM Staircase and a FEC expansion ratio of 2.5.



(b) LDGM Triangle vs. Staircase and a FEC expansion ratio of 2.5.



(c) RSE versus LDGM Staircase and a FEC expansion ratio of 1.5.



(d) LDGM Triangle vs. Staircase and a FEC expansion ratio of 1.5.

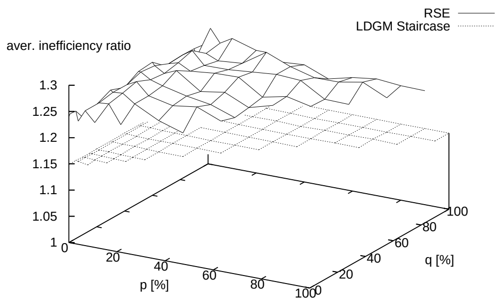
Figure 9.6: Tx_model_2.

9.3.2 Why is FEC Needed?

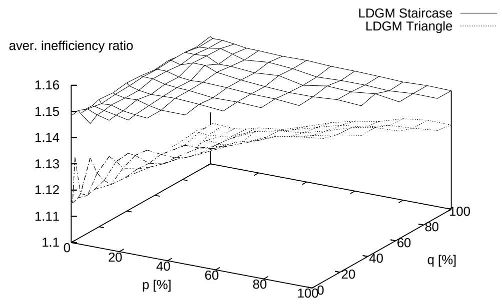
To motivate the use of FEC, we did a small test. Rather than using FEC to recover losses, a sender may decide to transmit each packet x times (remember that no back channel is available and therefore repeat request mechanisms are not an option in the context of this work). In figure 9.9 $x = 2$ and packets are sent in a random order. It shows that decoding is only possible with $p = 0$, and the average inefficiency ratio is then near 2.0 which means that the receiver waits almost systematically the end of the transmission to reconstruct the object. For all $p > 0$ at least one experiment failed, and therefore no inefficiency ratio is shown, as explained in section 9.3.1. This test highlights the *poor performances when using repetition only instead of FEC* for content broadcasting.

9.3.3 Tx_model_1: Send Source Packets Sequentially, Then Parity Packets

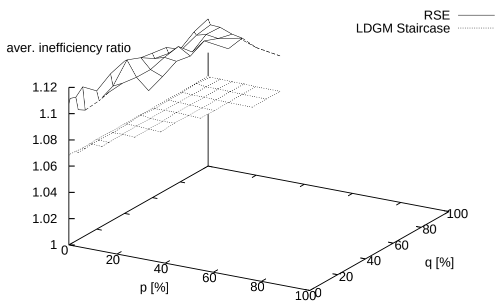
This scheme consists in sending all source packets sequentially, and then all parity packets, also sequentially. The results are shown in figure 9.5 (the LDGM Staircase curves are not shown since results are similar to LDGM Triangle).
We notice a first obvious result: without loss ($p = 0$) the inefficiency ratio is 1.0 with all codes. Indeed, all source packets are received and the receiver does not need any of the following parity packets. We'll see that other transmission schemes do not necessarily have this good property.
With losses (in burst or not) all codes show a similar behavior. The inefficiency ratio curve is very close to the $\frac{n_{received}}{k}$ curve for nearly all values of p and q. It means that the receiver always needs to



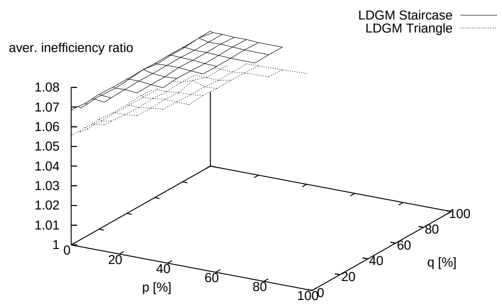
(a) RSE versus LDGM Staircase and a FEC expansion ratio of 2.5.



(b) LDGM Triangle vs. Staircase and a FEC expansion ratio of 2.5.

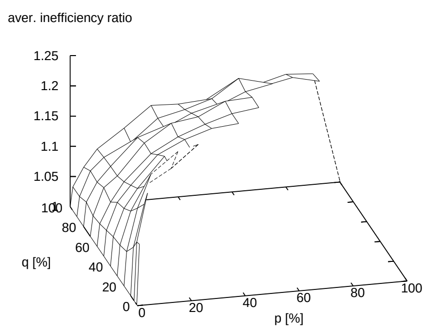


(c) RSE versus LDGM Staircase and a FEC expansion ratio of 1.5.

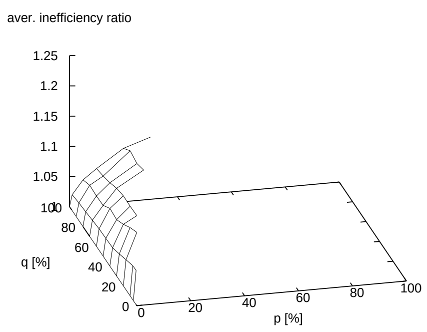


(d) LDGM Triangle vs. Staircase and a FEC expansion ratio of 1.5.

Figure 9.7: Tx_model_4



(a) RSE and a FEC expansion ratio of 2.5 (NB: the figure is rotated for better visibility).



(b) RSE and a FEC expansion ratio of 1.5 (NB: the figure is rotated for better visibility).

Figure 9.8: Tx_model_5

wait almost the end of transmission to reconstruct the object. With RSE this is not surprising since

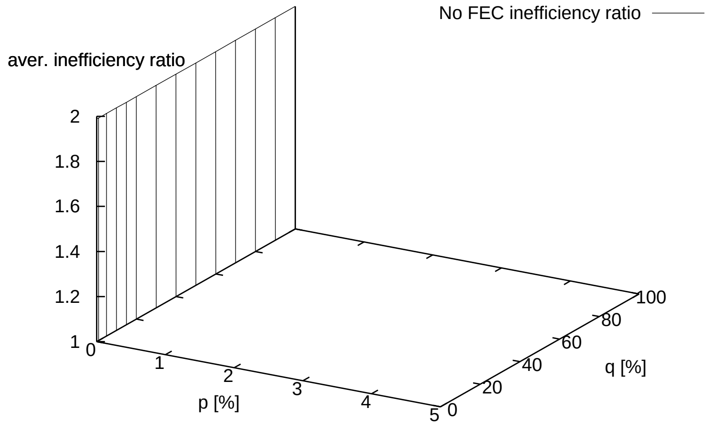


Figure 9.9: Performances without FEC but 2 repetitions.

the object is segmented in blocks: if a source packet of the last block is lost, the receiver needs to wait the end of the transmission, when the associated parity packets will be sent.

With LDGM codes this behavior is more surprising. These large block FEC codes encode the whole object directly, and parity packets are created using source packets coming from different parts of the content. However each parity packet depends on the previous one (because of the staircase in the parity check matrix, also present in LDGM Triangle codes). This dependency negatively impacts decoding performance when several sequential parity packets get lost.

Finally, RSE covers a smaller area (when $z - axis > 0$) than LDGM-* codes, which indicates that its erasure recovery capabilities are more limited. This is especially true for long packet loss bursts (small q). This is easy to understand: since packets are sent sequentially, and a single long burst results in the loss of a large number of packets of the same block. Decoding this block becomes difficult, especially with a small FEC expansion ratio (e.g. 1.5).

For all these reasons this transmission model is definitively bad, which was relatively foreseeable.

9.3.4 Tx_model_2: Send Source Packets Sequentially, Then Parity Packets Randomly

One idea to counteract the bad results of Tx_model_1 is to transmit the parity packets in a random order rather than sequentially. For RSE the advantage is clear: parity packets of the last few blocks can be transmitted earlier, so the receiver has on average less to wait before receiving parity packets for any block. This is confirmed in figures 9.6(a) and 9.6(c). The inefficiency ratio is much better than with Tx_model_1 and moreover this ratio is relatively constant.

LDGM Triangle codes exhibit relatively good performances and largely outperform RSE codes (figures 9.6(b) and 9.6(d)). Yet the LDGM Staircase 3D-curve has a hole around $p = 50$ and $q = 70$ (Figure 9.6(b)), where exactly one test failed. This is not acceptable in practice, and therefore Tx_model_2 is not recommended with LDGM Staircase and higher loss ratios. Conversely LDGM Staircase may be used with small loss ratios where the code yields very good results.

This transmission model confirms that with LDGM-* codes, parity packets should not be sent sequentially but randomly, to prevent the loss of sequences of parity packets. All the following tests will confirm this observation.

9.3.5 Tx_model_3: Send Parity Packets Sequentially, Then Source Packets Randomly

We now consider the case dual to Tx_model_2. We first transmit all parity packets sequentially. We can then transmit the source packets either sequentially or randomly. We only consider the latter case in this chapter since tests (not included in this work) have shown that sending the source packets sequentially yields uninteresting results (it makes no difference with LDGM-* codes, and we experience a worse behavior with RSE).
So called *non-systematic* codes, are codes that can decode using only parity packets (no source packet). RSE can be used as a non-systematic code if the number parity packets is high enough (i.e. $n-k \geq k$). This is not the case with LDGM Triangle and Staircase who need at least a small number of data packets to start decoding. Our tests have shown that with $p = 0$ the LDGM-* codes need exactly one source packet to decode the content, and therefore the inefficiency ratio is ≈ 1.5 for a FEC expansion ratio of 2.5.
With RSE, the receiver needs 29903 packets, that is to say decoding is possible when k packets of the last block have been received. Therefore the inefficiency ratio is ≈ 1.5 for FEC expansion ratios of 2.5.
Otherwise performances are not that interesting, and that transmission scheme may only be interesting for some specific loss patterns. Because of that the figures are not included in this chapter. The reader can look at [76].

9.3.6 Tx_model_4: Send Everything Randomly

In this transmission model, source and parity packets are sent in a fully random order.
Results are shown in figure 9.7. RSE offers the worst performances with an inefficiency ratio around 1.25. LDGM Staircase performs better and offers a ratio of 1.15. Finally LDGM Triangle yields the best results with ratios going from 1.12 to 1.14.
For the RSE and LDGM Staircase codes, the performances are relatively stable, independently of the packet loss behavior. This is not true with LDGM Triangle codes, that show better results with smaller p_{global}. This observation can generally be made when LDGM Triangle sends it parity packets randomly: it achieves better inefficiency ratios with smaller p_{global}. On the opposite, LDGM Staircase codes are not sensitive to p_{global} in this case.

9.3.7 Tx_model_5: Interleaving (RSE only)

Packet interleaving is a commonly used solution with small block FEC codes like RSE to increase their robustness against packet erasure bursts. The idea is to spread the transmission of one block over an interval that is longer than the loss burst duration. The maximum distance between two packet transmissions of the same block is achieved by sending successively one packet of each block, until reaching the last block, and then continuing with the following packet of each block, and so on. With LDGM codes this interleaving scheme is not feasible since there is only one block.
Results are shown in figure 9.8. As suspected, results are excellent with RSE, actually the best ones compared to all other transmission schemes. The packets are optimally aligned using interleaving, and for all loss patterns this solution give optimal results. This is not a surprise since interleaving has been intensively used along with RSE codes.

9.3.8 Tx_model_6: Send Randomly a Few Source Packets Plus Parity Packets

In this transmission model we transmit only a few source packets in addition to all parity packets. More precisely, we first pick randomly 20% source packets and schedule them randomly with all parity packets. This transmission model requires that the FEC expansion ratio be high enough (otherwise less than k packets will be received), and we chose 2.5.
The results are shown in figure 9.10. If all codes have constant performances, LDGM Staircase largely outperform other codes. Note that the fact that LDGM Staircase performs better than Triangle is rather unusual.

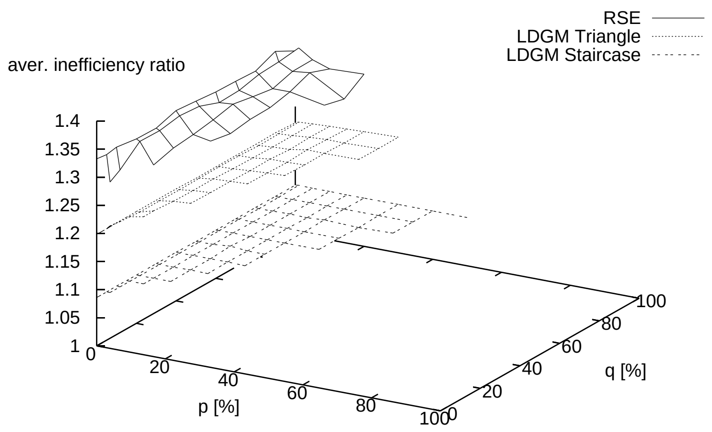


Figure 9.10: Tx_model_6 with LDGM Triangle, LDGM Staircase and RSE and a FEC expansion ratio of 2.5.

9.4 Simulation Results With a Reception Model

In this section we directly specify a reception model, without any consideration for the transmission and loss models that may generate it, i.e. there is no explicit transmission and loss model (no p and q) in that case. The goal is to better analyze an FEC code performances in a completely controlled environment.

9.4.1 Rx_model_1: Receive a Few Source Packets, Then Parity Packets Randomly

Section 9.3.8 has shown that sending only a few source packets along with the parity packets may be interesting. In this reception model, we further study this phenomenon. The difference with Tx_model_6 is that we now guarantee that these source packets arrive and are used for decoding. To do so the receiver first gets the source packets, and then, randomly, all parity packets.

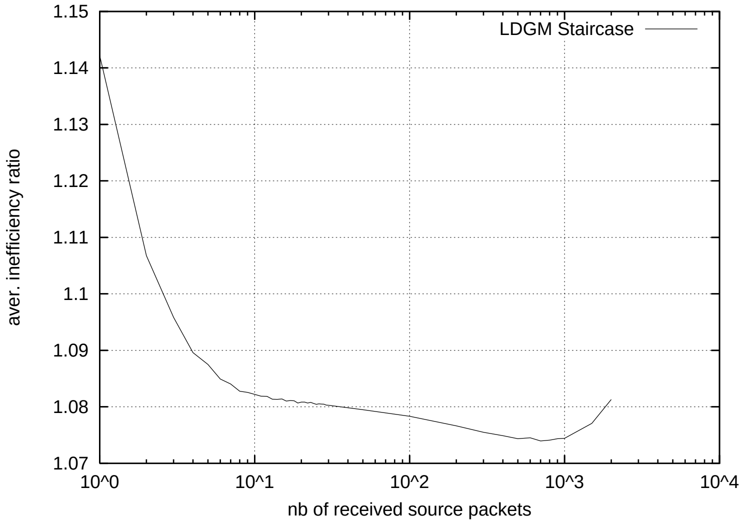


Figure 9.11: Rx_model_1 with LDGM Staircase.

We only consider LDGM Staircase and FEC expansion ratio of 2.5, since section 9.3.8 has shown that it yields the best results. We analyze the performance of the FEC codes as a function of the number of source packets received. Figure 9.11 shows that excellent performances are achieved when around 400 to 1000 source packets received. This is a very small number of packets compared to the object size ($k = 20000$ packets). Receiving more (or fewer) packets will degrade performances! To the best of our knowledge, we never saw such results mentioned in the literature, and we think this promising (and surprising) result deserves some complementary studies in future work.

9.5 Discussions and Recommendations

9.5.1 Summary of the Results

We now discuss the previous results and draw some recommendations. Regarding FEC codes performances:

- RSE: RSE should always be used along with Interleaving (which is not a new result). Yet, even in that case, RSE shows lower performances than the best LDGM codes, for instance LDGM Triangle with Tx_model_2 or Tx_model_4.
- LDGM Triangle and LDGM Staircase: In most cases (with some exceptions though, like with Tx_model_6), *LDGM Triangle yields better results.* For both codes it makes no difference whether source packets are sent randomly or sequentially. However sending parity packets sequentially *must to be avoided*, since loss bursts will severely degrade performances. It can also be observed that LDGM Triangle is often more sensitive to packet losses, probably because of higher dependencies between parity packets.

Regarding the transmission models:

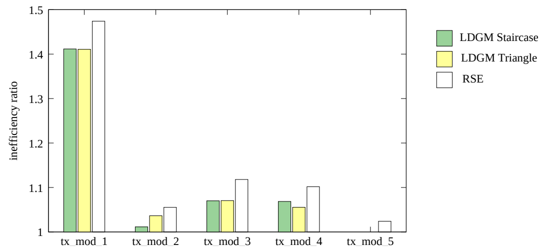
- Tx_model_1 and Tx_model_3 are of little or no interest in all cases.
- *Packet interleaving (Tx_model_5) is unavoidable with RSE, no matter the loss model.*
- The following models using totally or partially random transmissions have good results: sending sequentially source packets then randomly parity packets (Tx_model_2), sending everything randomly (Tx_model_4), and sending randomly a few source packets plus parity packets (Tx_model_6). More precisely:
  - Tx_model_2 is the preferred scheme for LDGM Triangle and LDGM Staircase if we want good results when *there are few packet losses.* With LDGM Staircase great care has to be taken of possible decoding failures with higher loss ratios.
  - Tx_model_4 (along with LDGM Triangle) and Tx_model_6 (along with LDGM Staircase) are the schemes that are the less dependent on the loss distribution. Therefore they are the *preferred solutions when the loss model is unknown, but Tx_model_4 will perform better with very high loss rates.*

9.5.2 In Practice

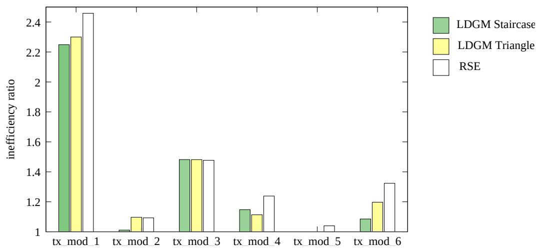
The previous results allow us to select an appropriate transmission scheme for a given use case. If the channel characteristics are unknown, LDGM Triangle and Tx_model_4 are excellent choices. If on the opposite the channel features (and its loss model) are known, we can compare all the FEC performances for all transmission schemes around the $(p;q)$ point, and make a choice.

The transmission can further be optimized by adapting n_{sent}, the number of packets actually sent. Remember that $n_{received} - n_{necessary\_for\_decoding}$ is the number of packets the receiver receives after decoding has finished. Our goal is now to have:

$$n_{received} = n_{necessary\_for\_decoding} + \epsilon \tag{9.2}$$



(a) FEC expansion ratio = 1.5



(b) FEC expansion ratio = 2.5

Figure 9.12: Example: Inefficiency ratios on the link Amher Massachusetts to Los Angeles.

($n_{received}$ should be a little bit greater since some tolerance is required [3]). By doing so, the number of packets received will be very close to the number of packets that are actually needed by a given receiver to successfully decode the object. This is achieved by reducing n_{sent} (Equation 9.1), that is to say by stopping transmissions after n_{sent} packets, without changing the scheduling. The last $n - n_{sent}$ packets will never been sent.
Note that selecting a smaller FEC expansion ratio to reduce the number of packets sent in front of a certain loss model, is not always feasible, especially when there are only a small number of predefined ratios possible (e.g. because the FEC codec has been optimized for some FEC expansion ratios). For instance, if only 1.5 and 2.5 ratios are possible, with RSE and Tx_model_5, a FEC expansion ratio of 2.5 is needed when $p = 40\%$. With 1.5, decoding would fail.
Besides we omit an essential performance metric: *encoding and decoding speed*. From this point of view, *LDGM codes are an order of magnitude faster than RSE codes*, as shown in the next chapter 10. This can be an essential criteria of choice when broadcasting big objects, for which the encoding and decoding times may be non negligible.
We now detail how to proceed in two specific use cases.

Homogeneous Receiver(s) and Known Channel

If there is only one receiver, or a set of receivers behind the same channel, the loss model may be identified and the p and q values determined (e.g. this can be the case when a content is broadcast in a cooperate network, or in some specific environments). Thanks to the p and q values, we can identify the best FEC code and transmission model for *this* environment.
Once they have been chosen, we can adapt n_{sent}. The inefficiency ratio is given by the associated curve. Then, according to formulas 9.1 and 9.2, the optimal n_{sent} is achieved when:

$$n_{sent} = \frac{n_{necessary\_for\_decoding}}{1 - p_{global}} \tag{9.3}$$

Let's consider the following example: a 50 MByte object is sent to a single receiver from Amhers Massachusetts to Los Angeles. [107] traced the loss behavior on this link and obtained $p = 0.0109$ and $q = 0.7915$ ($p_{global} = 0.0135$). Figure 9.12 shows the performances of the various FEC code and transmission models, for the FEC expansion ratios 1.5 and 2.5. We clearly see that *Tx_model_2 with LDGM Staircase and a FEC expansion ratio of* 1.5 give the best results ($inef\_ratio \approx 1.011$).
We now calculate the optimal n_{sent} (formula 9.3):

$$n_{sent} = \frac{1.011 * 50MBytes}{1 - 0.0135} \approx 51.24MBytes$$

With 1024 byte packet payloads, it means that the sender may stop transmitting after ≈ 50041 packets. In order to add some tolerance to this result we can decide to set n_{sent} to 55000 packets. This is significantly less than the $n = 73243$ packets that would have been sent otherwise, while preserving transmission reliability.

[3] To have an idea on the confidence interval around the average inefficiency ratios for the FEC codes used in this work please read [97] and chapter 10.

Heterogeneous Receivers and/or Unknown Channel

Imagine now that an object is broadcast over a wireless channel. All receivers observe different packet loss distributions, depending on many external parameters (e.g. movement, obstacles, distance to the source).
If the loss pattern of each receiver is known we may proceed as in the previous section, by choosing the (FEC code; transmission scheme; FEC expansion ratio) tuple that yields the best results for all receivers.
If the loss pattern of one or several receivers is unknown, which will probably be the general case, we need a universal scheme that shows the best possible performances in all situations.
Sending everything randomly (Tx_model_4) and sending only a few parity packets (Tx_model_6) are the schemes that are the less dependent of the loss pattern, and are therefore the preferred solutions in this context for LDGM codes. With RSE interleaving should be used, but performances will be lower.
As before n_{sent} can be adapted. We proceed as before with the main difference that we now have to look at the borders of the inefficiency ratio curve. The aim is to get $\frac{n_{received}}{k}$ as near as possible to the inefficiency ratio curve. This can be done by choosing some values of p and q that are located in the border region of the inefficiency ratio curve, evaluate the inefficiency ratio, and then proceed like in section 9.5.2.

9.6 Conclusions

This chapter investigated the impacts of packet scheduling and packet loss distributions on FEC performances, for content broadcasting applications. This work should be of great help for FLUTE-based file broadcasting systems, when there is no backward channel and where transmission reliability is achieved through the massive use of FEC and complementary techniques (e.g. cyclic transmissions within a carousel). Since we only consider file delivery systems, the transmission latency has little importance and large block FEC codes, like LDPC/LDGM codes, are good candidates.
The experiments carried out provide good insights on which (FEC code; FEC expansion ratio; packet transmission scheduling) tuples yield the best results for a given channel. Non surprisingly, experiments have shown that interleaving should always be used with RSE, no matter the loss pattern (which is not a new result). *Yet LDGM Staircase and Triangle codes usually perform significantly better than RSE.* For environments where the channel (and its packet loss distribution) is *unknown*, which is probably the general case, LDGM codes require a random transmission scheme: usually either (LDGM Triangle; Tx_model_4) or (LDGM Staircase; Tx_model_6). For environments where the loss distribution is *known*, our work helps to identify the best (FEC code; transmission scheme; FEC expansion ratio) tuple. Our results are therefore of utmost practical importance for optimizing broadcasting systems.
In the remainder of this thesis we will assume that the loss model is unknown and therefore almost all our experiments and simulation use a randomized transmission scheme. This chapter has shown that such transmission schemes yields the most constant results, independently of the loss pattern. This is important, since we are sure that our experiments and simulations produce valid results, applicable to a large set of use cases.
Future works will naturally consist in studying new transmission schemes and new FEC codes. More elaborated channel models may also prove to be useful for specific target environments. Other performance metrics will also be added, like the maximum memory requirements needed in each case.

72

Chapter 10

Performance Evaluation and Comparision of Four Large Block FEC Codes plus a Reed-Solomon Small Block FEC Codec

Contents

10.1 Introduction

The previous chapters presented the LDPC codes, its implementation and recommendation on its best usage. A detailed performance comparision and analysis of four Large Block FEC Codes LDGM, LDPC, LDGM Staircase and LDGM Triangle plus a Reed-Solomon Small Block FEC, based on the results of the previous chapters is presented now. We assume the usage of a randomized transmission, that, as we have seen in the previous chapter, yields the most stable results independently of the use case.
In this chapter we focus on:

- the global decoding inefficiency, caused either by the LDPC/LDGM FEC code or by the need to split the object into several blocks with RSE;
- the encoding and decoding speeds, LDGM, LDPC and RSE compared, which also determines the CPU load required by each of these codes, and hence their ability to be used either on high performance links or on lightweight hosts;

- and the memory requirements at both the encoder and decoder.

From these performance results, we identify the situations where each code operates the best, if any, and discuss some of their fundamental requirements and limits.
Note that memory requirements and encoding and decoding speeds have already been presented for LDGM Triangle and LDPC Staircase in chapter 8. In this chapter we focus on a comparision study with other FEC codes and therefore reuse the results of LDGM Triangle and LDGM Staircase.
A direct effect of patents on FEC codes is that, to the best of our knowledge, no public implementation of any Digital Fountain's codes exist, that could have been used for our performance evaluations. We therefore only consider the raw figures provided in [15, 88] without any possibility to explore some of the performance metrics or operational conditions that we believe are important but lack in these works.
The remainder of this chapter is structured as follows: we present the performance evaluation in section 10.2, starting with the decoding inefficiency, then the encoding and decoding times, and finally the memory requirements. Section 10.3 discusses the results obtained and compares them with related work. Finally section 10.4 concludes this chapter.

10.2 Performance Evaluation of the Four LDPC/LDGM FEC Codecs plus an RSE codec

10.2.1 Parameters and Performance Metrics Considered

The following parameters are considered during tests:

- *FEC expansion ratio, fec_ratio, and code rate:* the fec_ratio is the n/k ratio. The higher the ratio, the more parity packets are created. This FEC expansion ratio is also called "stretch factor" in [15]. This ratio is the inverse of the "code rate", defined as k/n and which is widely used in the coding community. FEC codes capable of producing an infinite number of parity packets have a rate equal to zero and are therefore called "rate-less". In the present work we will use the fec_ratio terminology.

- *object size:* in this work packets are always 1024 bytes long. We therefore express the object size in packets rather than in bytes. With LDPC/LDGM codes, the object being encoded as a single block, the object size in packets is equivalent to the k parameter of the FEC code. With RSE, the object is split into several blocks, each of size k packets.

We measured the following performance metrics:

- *global decoding inefficiency ratio, inef_ratio:* decoding inefficiency can be caused by:
  - the fact the FEC code is non MDS (e.g. LDPC/LDGM codes), so more than k packets must be received for the decoding a block of size k to complete (this is also known as the "overhead factor" in [88]), or
  - the need to split big objects into smaller blocks (e.g. RSE).

  In both cases $inef\_ratio * number\ of\ source\ packets$ packets, with $inef\_ratio \geq 1$, must be received in order to decode (all) the block(s). Said differently:

  $$inef\_ratio = \frac{number\ of\ pkts\ required for\ decoding}{number\ of\ source\ pkts}$$

  This *inef_ratio*, experimentally evaluated, is one of the key performance metrics we consider.

- *decoding inefficiency ratio confidence intervals:* considering the average *inef_ratio* only is not sufficient since large values are sometimes observed. We therefore measure various confidence intervals for values larger than the average, and the same for values smaller than the average.

- *encoding/decoding times and bandwidths:* As in chapter 8, we measure two types encoding of bandwidths: a global encoding bandwidth measured as:

$$global\_encoding\_bw = \frac{n * pkt\ size\ (in\ bits)}{time}$$

  and a partial encoding bandwidth which only takes into account the number of parity packets produced:

$$partial\_encoding\_bw = \frac{(n-k) * pkt\ size\ (in\ bits)}{time}$$

  The first measure indicates whether the codec can support a given transmission rate when using a systematic FEC code (the default, both source and parity packets are sent), whereas the second one refers to what is actually *produced* by the FEC codec.

  No such distinction exists at the receiver, and the bandwidth is always measured as:

$$\frac{number\ of\ pkts\ required for\ decoding * pkt\ size\ (in\ bits)}{time}$$

  Since this value depends on the global decoding inefficiency ratio which varies slightly at each test, we only consider average values.

- *maximum memory requirements:* we measure the maximum memory requirements, both during encoding and decoding, caused by the need to store source packets (and often parity packets), but also all the codec's internal buffers or tables, and in particular the parity check matrix with LDGM/LDPC codes.

10.2.2 Experimental Setup

Testing Application

The following tests use both our LDPC/LDGM codec and a widely used RSE codec [91]. We use the same testing application as in chapter 8. Given an object and a fec_ratio, the application first performs FEC encoding: to k source packets of a block, the codec produces $n-k = (fec\_ratio-1)*k$ parity packets. Both source and parity packets are 1024 byte long. Once encoding has been performed, the source and parity packets (of all blocks with RSE) are transmitted in a fully random order. The receiving application waits until it has received enough packets to decode the block (or all blocks with RSE). He then stops reception and reports the number of packets received and the global $inef\_ratio$ measured.
Several remarks can be made regarding this methodology:

- There is no explicit loss simulation model (e.g. random or per burst) because random transmissions are not affected by the loss model. Performance results could be slightly different if other transmission models were used, for instance by sending all k packets in sequence except a few of them (assumed erased) and then parity packets. As we have shown in the previous chapter this methodology has a major drawback: it introduces additional parameters like the loss model and the loss ratio(s). On the contrary our approach avoids these problems, is more universal, and is in line with the way these codes can be used for instance in ALC sessions [32, 95, 96].

- The same application and environment is used for testing all codes, including RSE. It makes comparisons trustworthy.

- No transmissions on the network are performed, and the same application performs both encoding and decoding. This feature enables us to better control the way packets are "sent", simplifies tests, reduces the overall test durations and enables accurate encoding/decoding time measurements.

Determining Each Block Size with RSE

Because of the intrinsic limitations of a RSE codec working on $GF(8)$, a large object has to be fragmented into several blocks compatible with the codec limitations: $k \leq n \leq max\_n = 256$. Determining the block structure of an object requires two things. The first key parameter is the maximum source block size possible, max_k. Two strategies exist for that:

- for a given FEC expansion ratio, we compute: $max\_k = max\_n/fec\_ratio = 256/fec\_ratio$. Doing so leads to an optimal use of the RSE codec, since we are sure to always operate with blocks of maximum size.
- or we set max_k to a fixed value and produce the required $n - max\_k$ parity packets, as long as $n \leq max\_n$ (e.g. we can have a FEC ratio of at most 5 if $max\_k = 51$). This scheme does not enable an optimal use of the RSE codec, since larger blocks could often be possible, but a fixed, smaller max_k is sometimes chosen because of performance issues (see table 10.1).

The second key feature is the blocking algorithm (see section 3.2.2). We chose the one specified in the FLUTE standard [82] which is known to be optimal since this scheme avoids situations where the final block is much smaller than all previous maximum size blocks, a situation largely sub-optimal (an extreme case is a last block of size one packet only). Here all blocks are of the same size, $A\_large \leq max\_k$, or one packet less than A_large.
Taking care of these two aspects is of high importance for reliably comparing the performances of the RSE and LDPC/LDGM codes, and we consider both ways of determining max_k in our tests.

Tests Carried Out

We conducted two kinds of tests:

- *Fixed FEC expansion ratio and different object sizes:* the fec_ratio is set to 1.5, while the object size varies from 100 packets (100 KBytes) up to 50000 packets (≈50 MBytes) [1].
- *Fixed object size and different FEC expansion ratios:* the object size is set to 20000 packets (≈20 MBytes), while the fec_ratio varies from 1.25 up to 5.0 (or, equivalently, the code rate varies from 0.8 down to 0.2).

During these experiments we measure the minimum number of packets required for decoding, and the encoding/decoding times. Experiments are repeated 1000 times with LDGM, LDGM Staircase, LDGM Triangle and RSE. We limit the number of LDPC experiments to 100 only due to performance issues (see section 10.2.4). The average value, and the 99% confidence interval for the values lower or higher than this average, are reported. We also discuss the absolute minimum/maximum values and higher confidence intervals (99.9% and 99.99%) in section 10.2.3.
Note that our transmission scheme leads to a *maximum decoding inefficiency equal to* $n/k = fec\_ratio$ since the number of packets required for decoding is at most equal to the number of packets transmitted, $n = fec\_ratio * k$.

10.2.3 Decoding Inefficiency Experiments

Fixed FEC ratio and different object sizes

We first compare the results using a fixed FEC ratio. Results are given in figure 10.1. We see that LDGM Triangle obtains the best results, with an inefficiency ratio that quickly stabilizes around 1.055. The 99% confidence interval is quite small, $[1.051; 1.059]$, with an object of size $50,000$ packets.
Like LDGM Triangle, the inefficiency ratios of LDGM Staircase and LDPC decrease and quickly stabilize when the object size increases. At the same time the confidence interval gets smaller. LDGM Staircase is a bit less efficient than its triangle variant, with an inefficiency ratio around 1.068 and a 99% confidence interval of $[1.064; 1.071]$ with an object of size $50,000$ packets. LDPC's inefficiency ratio is a bit higher, around 1.076, with a 99% confidence interval of $[1.073; 1.079]$.

[1] Note that unlike [88] we do not consider small objects, since small block FEC codes like RSE are in that case preferable to large block codes.

LDGM behaves differently than the other three codes: its inefficiency ratio is much worse and increases slowly with the object size (a result already observed in our previous work [94]), the confidence intervals are larger, and the overall performances depend on the left degree parameter. 7 is the optimal degree in this case [94], and the LDGM inefficiency ratio is around 1.155 with an object of size $50,000$ packets. But changing the FEC expansion ratio also changes this optimal left degree. No such dependency exists with the other large block FEC codes where a left degree of 3 always yields the best results, which is in line with the results mentioned in the literature for LDPC.
Finally figure 10.1 (e) and (f) shows that RSE performances depend on the object size: the larger the object, the lower the efficiency, which is not a surprise (section 2.1.2). This has to be compared with the constant decoding inefficiency observed with LDGM Staircase/Triangle codecs[2]. With an object of size $50,000$ packets, RSE has a decoding inefficiency around 1.122 with fixed n, and 1.220 with fixed k, which is far behind LDGM Triangle. Besides the confidence intervals are much higher with RSE. On the opposite, with very small objects, RSE remains the best choice, essentially because it's an MDS code.

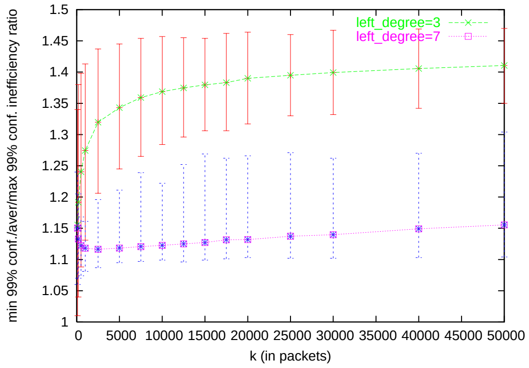
Fixed object size and different FEC ratios

The behavior of the four large block FEC codes is much more differentiated if we vary the FEC expansion ratio. This is visible in figure 10.2. First we notice that the LDPC curve stops at a FEC ratio of 3.0. This is due to a pure practical reason and shows a major limitation of the LDPC codec used: LDPC has a very high memory consumption, and above a FEC ratio of 3.0 the required memory exceeds the 2GB RAM of our machine. Moreover we see that even for FEC ratios smaller than 3.0, LDPC performances are not quite interesting, even if the confidence interval is small.
LDGM with a left degree equal to 10 has also poor performances, with an inefficiency ratio of 3.045 when the FEC expansion ratio is 5.0.
The LDGM Staircase and Triangle codes yield the most interesting results. *LDGM Triangle is the code that performs the best for FEC ratios smaller than 2.5.* At 2.5, LDGM Triangle has an average inefficiency ratio of 1.115 and a confidence interval going from 1.103 to 1.127. LDGM Staircase has an inefficiency ratio of 1.148 and a 99% confidence interval of $[1.138; 1.159]$ for the same FEC ratio.
For FEC ratios greater than 2.5, the performance of LDGM Triangle decreases suddenly. *On the opposite LDGM Staircase performs well for FEC ratios larger than 2.5.* LDGM Staircase reaches a local minimum at a FEC ratio of 2.75 (inefficiency ratio of 1.138) and behaves very well up to a FEC ratio of 5.0 (inefficiency ratio of 1.325, compared to 1.849 with LDGM Triangle), with a small confidence interval, $[1.314; 1.338]$.
Finally this kind of tests is more favorable to RSE than the previous ones with a fixed FEC expansion ratio, essentially because RSE is an MDS code (indeed, if there is a single block, producing more parity packets does not lead to performance degradation with an MDS code).
Yet, with the fixed n strategy (figure 10.2 (e)), the block size decreases if the FEC ratio increases (let's recall that $max\_k = 256/fec\_ratio$), there are more and more blocks and the global inefficiency ratio also increases. The inefficiency ratio is 1.241 with a FEC ratio of 2.5, and 1.512 with a FEC ratio of 5.0, which is worse than LDGM Staircase in both cases, and higher than LDGM Triangle in the first case. With a fixed k (figure 10.2 (f)) we observe that the global inefficiency ratio is worse with small FEC ratios, but about the same than with the fixed n strategy for higher FEC ratios (this is normal since the max_k value is about the same if $fec\_ratio = 5.0$). We also see that the confidence interval is rather large with RSE and increases with the FEC ratio, whereas no such behavior exists with other codes (except LDGM).
To conclude, RSE is not the best choice which remains LDGM Staircase for high FEC ratio values, and LDGM Triangle for small FEC ratio values.

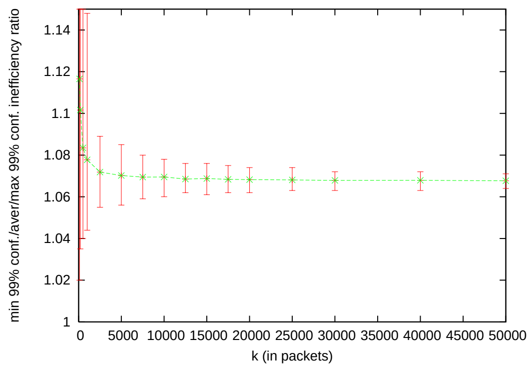
Variations in the Results

A result not highlighted on figures 10.1 and 10.2, is that the absolute maximum values (worst global inefficiency ratio) can largely exceed the average, even if this is extremely rare (less than 1%, since we generated these graphs with a confidence interval of 99%). These peaks appear with all three

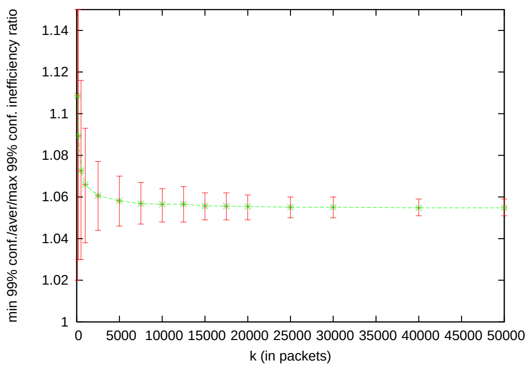
[2] At least as long as the maximum block size of such codes is not achieved, which essentially depends on the available memory size. Above this threshold, several blocks must be used, each of them being independently encoded.



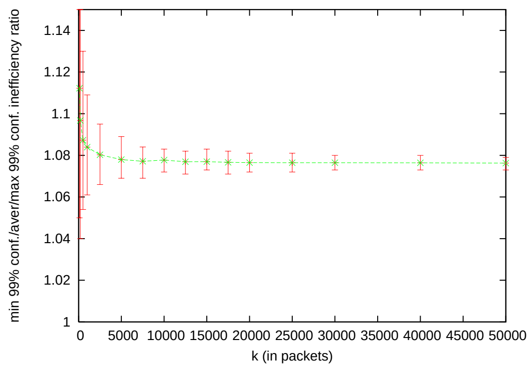
(a) Inefficiency ratio of LDGM



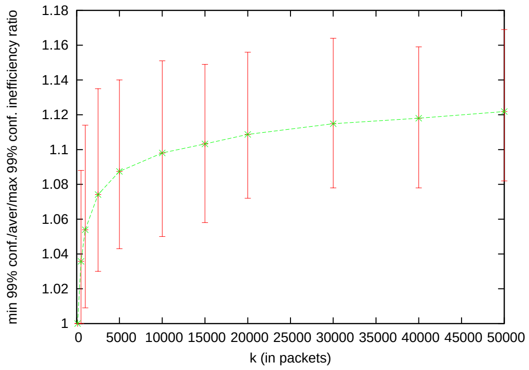
(b) Inefficiency ratio of LDGM Staircase



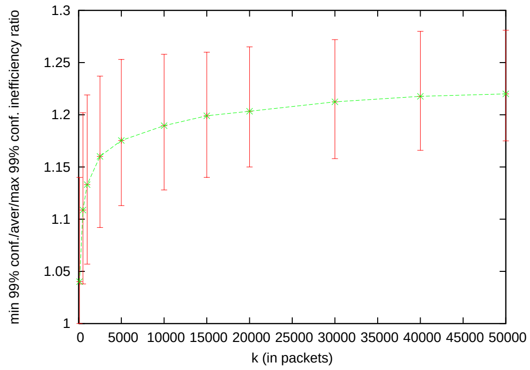
(c) Inefficiency ratio of LDGM Triangle



(d) Inefficiency ratio of LDPC



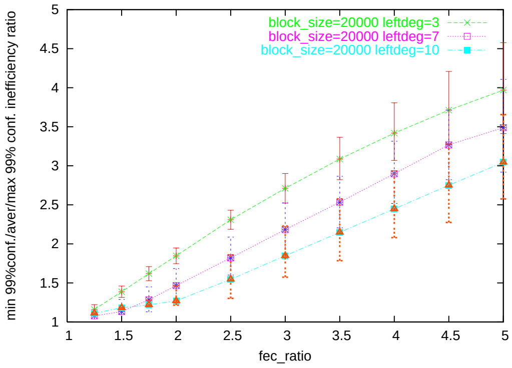
(e) Inefficiency ratio of RSE with fixed n(=256).



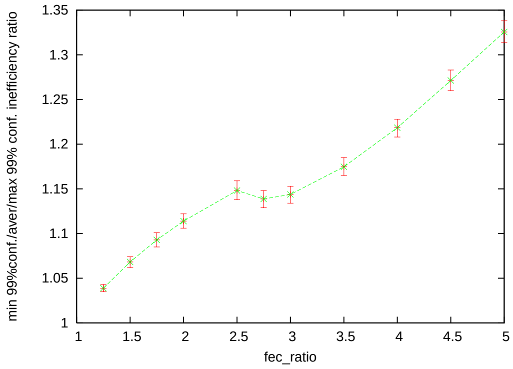
(f) Inefficiency ratio of RSE with fixed k(=51).

Figure 10.1: Global inefficiency ratio of LDPC/LDGM/RSE codes as a function of the object size, with FEC ratio = 1.5.

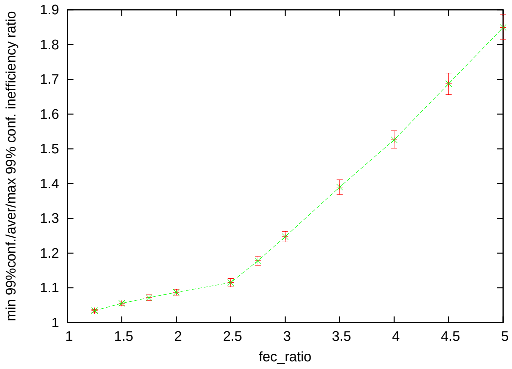
LDGM codecs, but their number and importance decreases with the block size. We did not notice this behavior with LDPC though, but this can be due to the fact we only did 100 tests, not 1000. No such behavior happen for the absolute minimum values (best inefficiency ratio), unfortunately. In order to quantify this behavior more precisely, we carried out other experiments with the LDGM Staircase and Triangle codes, with a fixed object size of 20, 000 packets and a FEC expansion ratio of



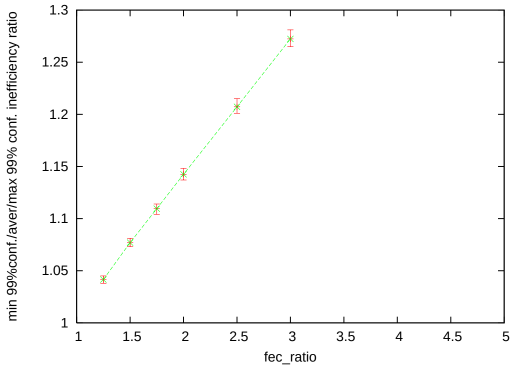
(a) Inefficiency ratio of LDGM



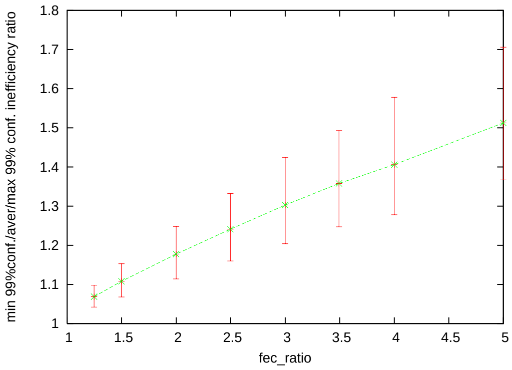
(b) Inefficiency ratio of LDGM Staircase



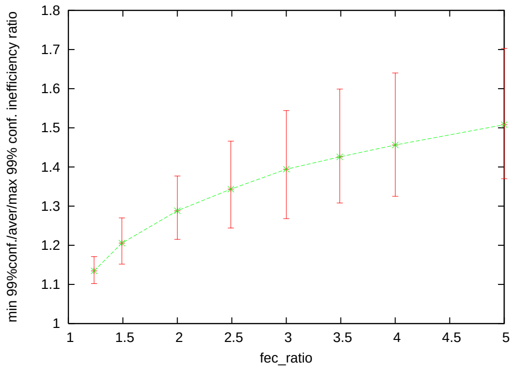
(c) Inefficiency ratio of LDGM Triangle



(d) Inefficiency ratio of LDPC



(e) Inefficiency ratio of RSE with fixed n(=256).



(f) Inefficiency ratio of RSE with fixed k(=51).

Figure 10.2: Global inefficiency ratio of LDPC/LDGM/RSE codes as a function of the FEC ratio, with object size = 20000 packets.

1.5, repeating the tests 100,000 times. Results are shown in figure 10.4.

We can see that even with a confidence interval of 99.9% we are quite close to the mean. Yet, by increasing the confidence to 99.99%, the upper confidence interval bound starts to become important (1.187 for LDGM Triangle). The absolute maximum is 1.405 for LDGM Triangle and 1.348 for LDGM Staircase. In spite of small differences, LDGM Staircase and LDGM Triangle show the same global

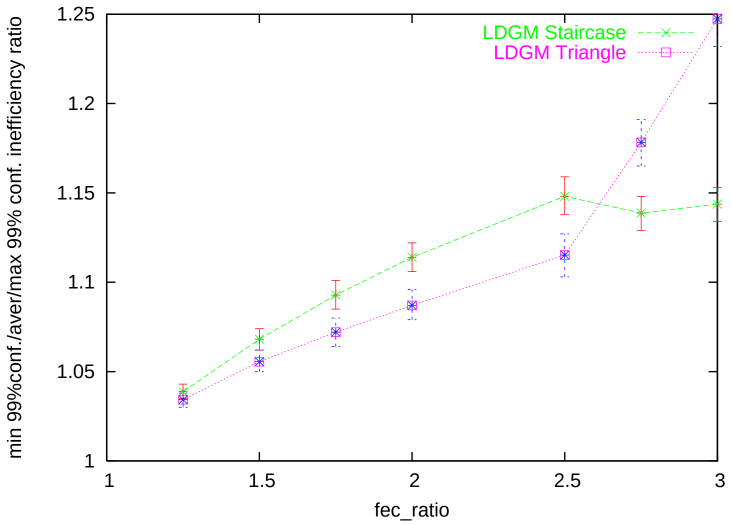


Figure 10.3: Global inefficiency ratio of LDGM Staircase and LDGM Triangle codes as a function of the FEC ratio (closeup).

behavior.
We cannot give an explanation for these peaks yet. This will be considered in future works.

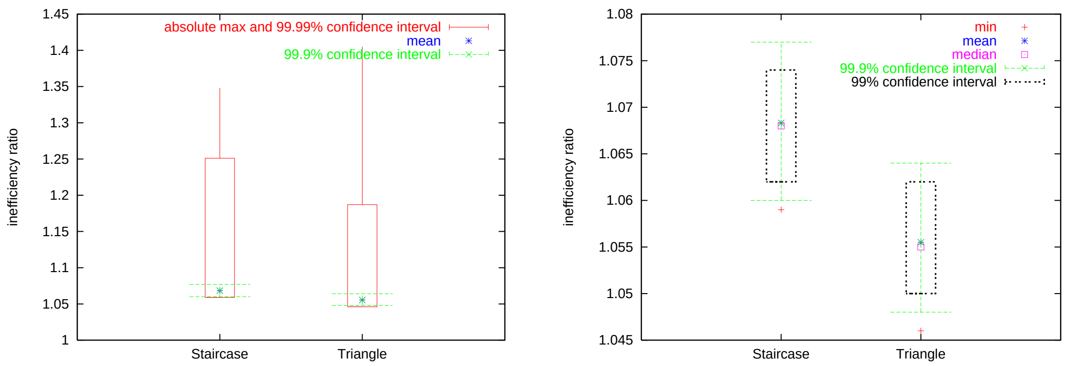
10.2.4 Encoding/Decoding Time Experiments

We now evaluate the encoding/decoding times and the associated bandwidths of all codes. Note that the experiments carried out here use the optimized algorithms presented in chapter 8.

Encoding/Decoding Times with a Fixed Object Size and Fixed FEC Ratio

As already pointed out, the encoding times for LDPC are very high. For instance the encoding of a $20,000$ packet object with a FEC ratio of 1.5 requires 281 seconds on a Pentium IV 3.06 GHz machine (figure 10.5). Note that our implementation is absolutely not optimized for LDPC encoding, and major improvements will probably be observed by using one of the optimizations suggested in the literature. But in all cases, the performances will remain behind that of the various LDGM codes, since these latter enable an immediate encoding, which will never be possible with LDPC.
The decoding time is also very high (175 seconds) because our LDPC codec requires that the G generator matrix be produced at the receiver too for some internal technical reason[3]. This limitation could be removed in the future, and decoding would then be similar to that observed with all LDGM codes, since the same algorithm is used in all cases. Producing graph 10.1 (d) took more than a week on a Pentium IV 3.06 GHz with 2GB RAM, whereas tests were repeated only 100 times (compared to 1000 times with LDGM codes). The resulting encoding bandwidth (Table 10.1) is low, less than 1Mbps.
In comparison all LDGM codes are very fast (e.g. producing graphs 10.1 (a), (b) and (c) took only one night). There are differences shown in figure 10.5, essentially caused by the density of the H parity check matrix, which determines the number of XOR sums required to produce each parity packet. For instance, with LDGM Staircase, a parity packet is the sum of only $right\_degree_{LDGM\_Staircase} - 1 = 1 + (3 * k)/(n - k)$ packets with LDGM Staircase, whereas it is the sum of $right\_degree_{LDGM} - 1 = (7 * k)/(n - k)$ packets with LDGM. In all cases, the resulting encoding speeds are suitable for transmissions over high speed networks.
Decoding is a bit longer (cf. figure 10.6) since more manipulations are required (like searching all constraint equations where a given packet takes part and updating the value of the partial sum).

[3]Matrix columns are permuted by the encoder in order to produce the G matrix, and this permutation must be reproduced at the decoder, which is only possible if G is produced.



(a) average/max values and 99.99%/99.9% confidence intervals

(b) average/median/min values and 99.9%/99% confidence intervals (close-up of previous figure)

Figure 10.4: Variations of inefficiency ratios with LDGM Staircase/Triangle, with object size = 20000 packets and FEC ratio = 1.5.

However the resulting speeds are also sufficient for high speed network transmissions as show in table 10.1.
Figure 10.5 and 10.6 show that the decoding and especially encoding times for the two RSE variants are significantly slower: LDGM Staircase is about 29.81 times faster during encoding and 13.72 times faster during decoding than RSE/fixed $n = 256$, and respectively 7.44 and 2.96 times faster than RSE/fixed $k = 51$ (table 10.1). This may become a major issue when working with constrained devices (e.g. PDA), where battery and CPU power is limited.

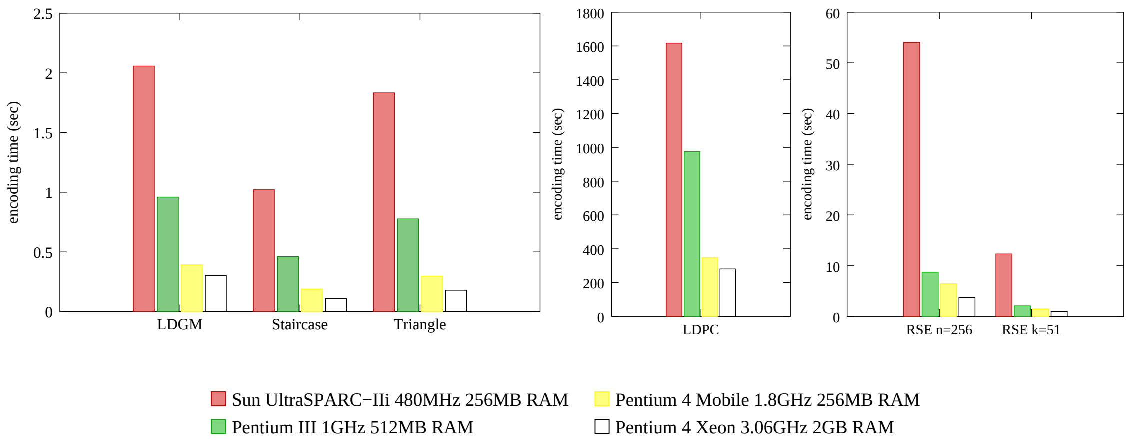


Figure 10.5: Average initialization plus encoding times for all codes, with object size = 20000 packets and FEC ratio = 1.5.

Encoding/Decoding Times with Higher FEC Ratios

We now analyze the decoding and encoding times as a function of the FEC expansion ratio, for a fixed object size of 20,000 packets (figure 10.7).
The encoding times (figure 10.7 (a) and (c)) yield no surprising results. LDGM Staircase remains the fastest codec even with high FEC ratios. The curves remain linear with LDGM Staircase and LDGM Triangle, and the encoding speeds remain acceptable.
Compared to LDGM Staircase, RSE remains quite slow. It is worth noting that RSE/fixed k has

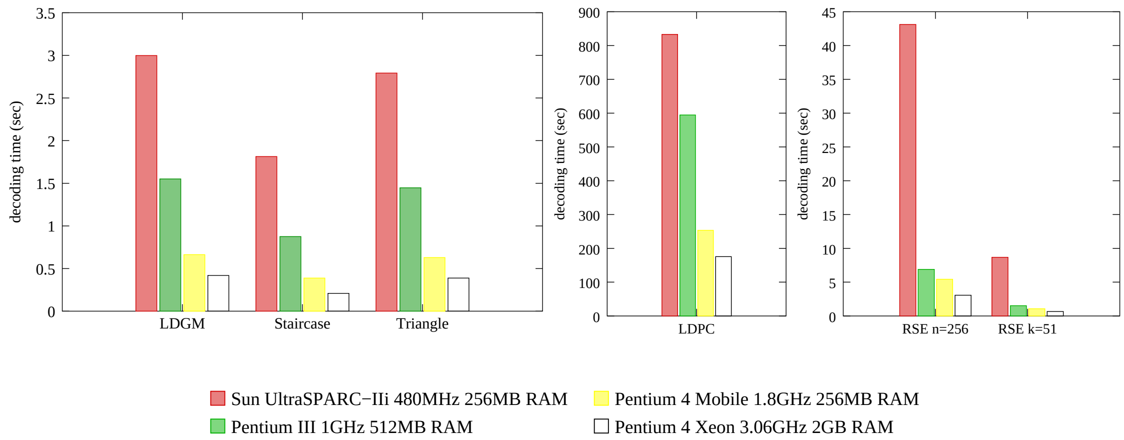


Figure 10.6: Average initialization plus decoding times for all codes, with object size = 20000 packets and FEC ratio = 1.5.

Code	*Encoding, produced FEC*	*Encoding, produced data + FEC*	*Decoding*
LDGM (left deg. 7)	264.024 Mbps	792.080 Mbps	432.320 Mbps
LDGM Staircase	734.998 Mbps	2,204.990 Mbps	816.505 Mbps
LDGM Triangle	443.813 Mbps	1,331.440 Mbps	434.866 Mbps
LDPC	0.288 Mbps	0.856 Mbps	0.982 Mbps
RSE (n=256)	21.471 Mbps	64.412 Mbps	52.255 Mbps
RSE (k=51)	86.00 Mbps	257.99 Mbps	241.97 Mbps

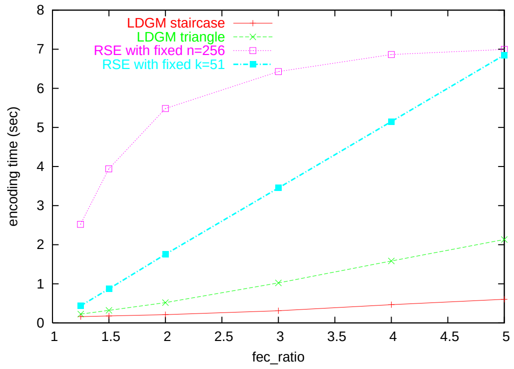
Table 10.1: Average encoding/decoding speeds on a Pentium 4 Xeon 3.06GHz 2GB RAM, with object size = 20000 packets and FEC ratio = 1.5.

a *linear encoding time curve as the object size increases*, unlike common belief. Some results like [15] that show a quadratic encoding time increase, assume that the whole object is encoded as a single block, which requires operating on GF(16) for instance, which is known to yield prohibitive performances, and in no case correspond to the way an RSE codec is used in most applications.
Finally, the LDPC performances (figure 10.7 (c)) turn out to be catastrophic with higher FEC ratios.
At the decoder (figure 10.7 (b), (d)), the results are as expected for LDGM Staircase, Triangle and LDPC. LDGM Staircase and Triangle show comparable performances: they are very fast, with a slight advantage for LDGM staircase, and decoding time increases linearly. LDPC show about the same behavior at the decoder than at the encoder because of the limitation already mentioned in our implementation.
The surprise comes from RSE with a fixed n: decoding time remains reasonable, and even decreases after a maximum at a FEC ratio of 2.0. RSE even becomes faster than LDGM triangle for FEC ratios above ≈ 4.5. The explanation comes from the fact that the block size k of the RSE code is calculated from the maximum value of n divided by the FEC ratio. This means that n decreases with increasing FEC ratio. The number of block increases in the same time, but the time gained having small blocks is more important than the time lost having a higher number of blocks. We also see that with a fixed k decoding times increases less rapidly than with LDGM triangle. At a FEC ratio of ≈ 3.3 they have the same decoding time.

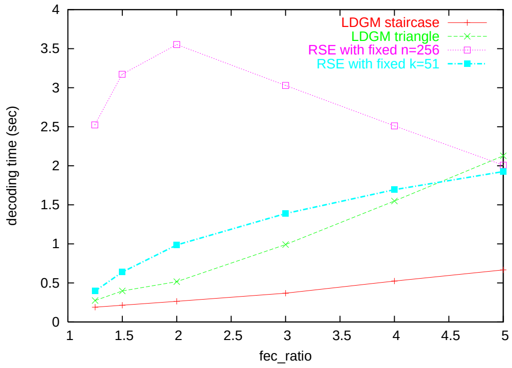
10.2.5 Memory Requirements

We now analyze the *maximum* memory requirements for the different codecs, both from the encoder and decoder applications points of view. We further distinguish:

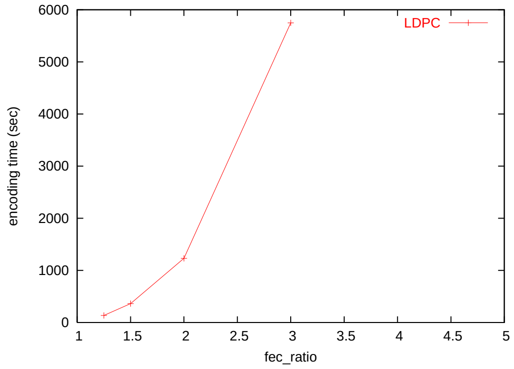
- the memory storage of the H matrix (for all LDPC/LDGM codes), plus G in case of LDPC.



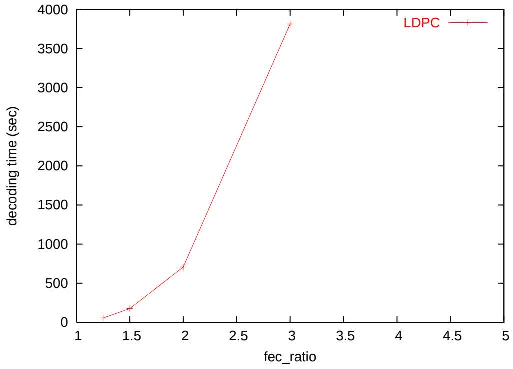
(a) LDGM Staircase/Triangle and RSE encoding times



(b) LDGM Staircase/Triangle and RSE decoding times



(c) LDPC encoding times



(d) LDPC decoding times

Figure 10.7: Average encoding/decoding times as a function of the FEC expansion ratio on a Pentium 4 Xeon 3.06GHz 2GB RAM, with a fixed object size = 20000 packets.

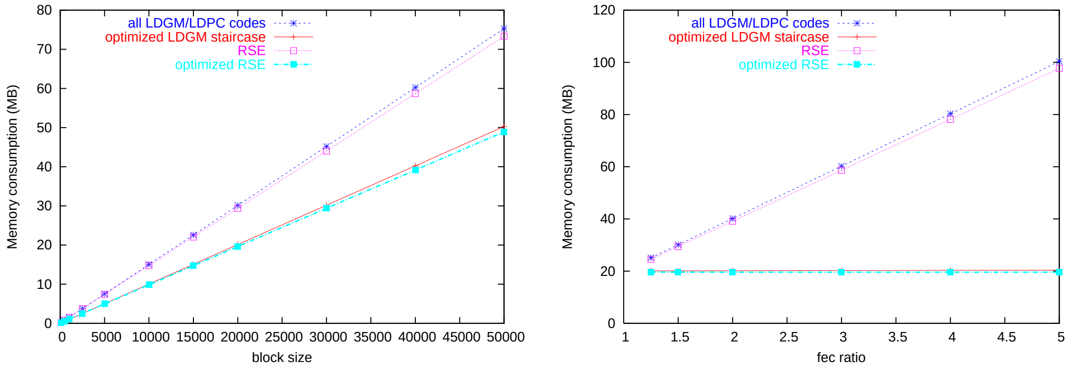
- all the remaining memory requirements, essentially for the source and parity packets (except in optimized versions, see below), the $n-k$ constraint nodes for a decoder (with LDPC/LDGM codes), and also the various buffers or tables required by the codec.

We also consider optimized versions, from a memory requirement point of view, of the encoding or decoding applications:

- an optimized encoding application produces a parity packet on the fly, uses it immediately (e.g. by sending it), and then frees the associated packet buffer. This is possible with RSE and LDGM/LDGM Staircase codes, but not with LDGM Triangle where dependencies exist with previously created parity packets.

- in an optimized RSE decoding application, each time a block has been decoded and consumed by the application, the buffers containing the associated packets are deallocated. This optimization assumes that the application can manage each block independently of others, for instance by storing their content on a disk in case of a file transfer [95, 96].

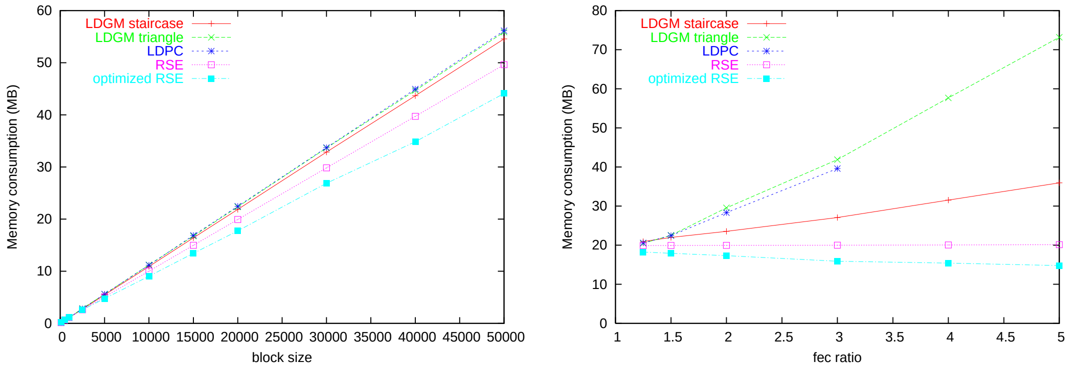
Figures 10.8 (a) and (b) show the memory requirements at the encoder, not considering the H/G matrices. We see that all non optimized applications (and codecs) have the same requirements, essentially dominated by the amount of data required to store the source and parity packets. The two RSE variants, with fixed n and with fixed k, also have the same memory requirements and are therefore represented with only one curve.



(a) All codes, with a fixed FEC ratio = 1.5.

(b) All codes, with a fixed object size = 20000 packets.

Figure 10.8: Memory consumption (without H/G matrices) during encoding.



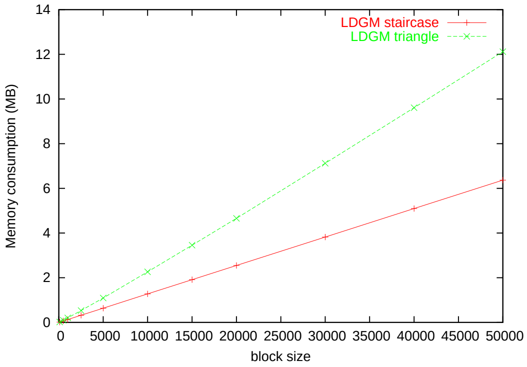
(a) All codes, with a fixed FEC ratio = 1.5.

(b) All codes, with a fixed object size = 20000 packets.

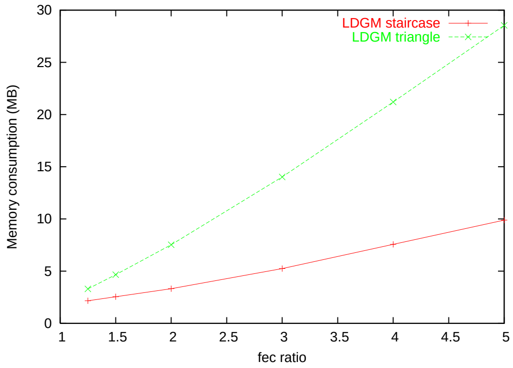
Figure 10.9: Memory consumption (without H/G matrices) during decoding.

The situation is quite different when considering decoding (figures 10.9 (a) and (b)). Here RSE (both with fixed n and fixed k) offers significant memory savings, especially with large FEC ratios, even with non optimized applications. The reason is the following: increasing the FEC ratio with RSE does not lead the decoder to store more packets, since a maximum of $k = A\_large$ (or $A\_large - 1$) packets are required for each block. The two LDGM-* codes uses the optimized decoding algorithm of chapter 8, where we already explained how when and how source and parity packets are stored in memory at the decoder.

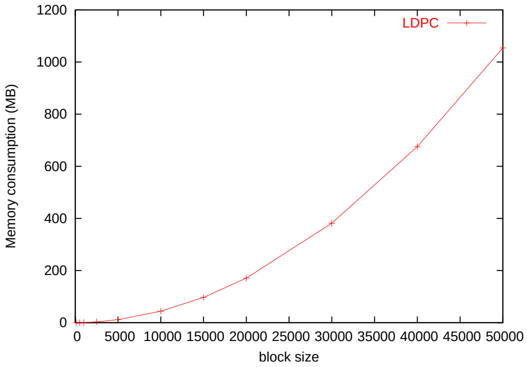
The memory consumption for the H (plus G with LDPC) matrices is shown in figures 10.10 (a) to (d). We see that LDPC needs a prohibitive amount of memory for matrix G: 171.12 MB with a block size of $20,000$ packets and a FEC expansion ratio of 1.5. This must be compared to the 2.5 MB required for LDGM Staircase and 4.6 MB for LDGM Triangle for the same parameters. The requirements are higher when the FEC expansion ratio increases, especially with LDGM Triangle which requires 28.5 MB with a FEC ratio of 5.0, whereas an LDGM Staircase “only” requires 9.9 MB in that case. Therefore the memory requirements for storing the H matrix, which largely reflect the density of the matrix, is not a major issue with LDGM codes for small to medium FEC expansion ratios, and remains low in front of the memory needed to store packets during encoding.



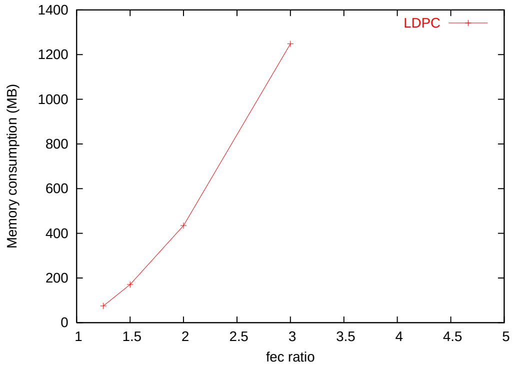
(a) LDGM Staircase/Triangle codes, with a fixed FEC ratio = 1.5.



(b) LDGM Staircase/Triangle codes, with a fixed object size = 20000 packets.



(c) LDPC with a fixed FEC ratio = 1.5.



(d) LDPC with a fixed object size = 20000 packets

Figure 10.10: Memory consumption for the H matrix (plus G for LDPC).

10.3 Discussion and Related Works

10.3.1 Which Code to Use, and When?

The previous performance results clearly define the main field of applications of each code, which, except for LDPC and LDGM, are rather complementary:

- *Regular LDPC showed prohibitive memory requirements and encoding speed*, even if the situation is much better at the decoder[4]. High FEC ratios (> 3.0) could not be supported even on very powerful machines. The measured inefficiency ratio is comparable (but worse) to those obtained with LDGM Staircase/Triangle. We therefore do not recommend the use LDPC which offers no advantage at all.

- *LDGM* is rather limited. It is a fast code but the inefficiency ratios obtained are not satisfying, especially with higher FEC ratios. We therefore do not recommend its use.

- *LDGM Staircase is the best code for FEC expansion ratios* > 2.5 where it outperforms all others codes. In all cases it offers a high encoding/decoding speed (it's the fastest code we tested), along with a low and stable (an important feature) global inefficiency inefficiency ratio.

[4]We ignore the limitation mentioned in section 10.2.4 since this can probably be solved –even if we did not try– in an optimized implementation.

- *LDGM Triangle is the best code for FEC expansion ratios* ≤ 2.5. In this case it outperforms LDGM Staircase in terms of inefficiency ratio (here also fairly stable) while keeping a good (but lower) encoding/decoding speed. But performances quickly decrease with a FEC ratio > 2.5, in which case the Staircase variant is preferable.
- *RSE* remains an excellent choice for small objects, or when the maximum memory requirements at the decoding side must be kept low, especially with large FEC expansion ratios. But its major limitations are (1) its high encoding (and often decoding) time (and CPU load) compared to LDGM Staircase/Triangle codes, an issue with lightweight hosts, and (2) a global inefficiency ratio much lower than those achieved with LDGM Staircase/Triangle codes.

With all four large block FEC codes, decoding is always fast, but less than encoding, because of more complex operations involved that require to cross the H matrix and store intermediate sum values in check nodes. The performance level is nevertheless compatible with high transmission rates, in the order of several hundreds of Mbps.

10.3.2 Comparison With Other Fixed-Rate Codes

We now give some elements of comparison with several fixed rate codes. [88] gives recent performance evaluations of various codes derived from LDPC: Tornado-like codes, LDPC codes and IRA codes. In each case the authors only consider patented irregular graphs. Because many different degree distributions are possible, the authors systematically test all distributions mentioned in the literature (80 such distributions are considered), and only report the best results. A comparison with our results highlights that:

- using irregular distributions is for sure effective, on condition the right distribution is used. For instance, with a $20,000$ object and an FEC expansion ratio of 1.5, the authors achieve an approximate 1.02 decoding inefficiency with all codes, whereas our best code only achieves 1.055.
- the results obtained with irregular distributions still improve when the object size increases, falling below 1.01 with IRA codes and objects of size $100,000$ packets, while in our case an equilibrium is achieved around 1.055.

In [15], using the Tornado Z codec, the authors mention a decoding inefficiency on the average of 1.054 with a FEC expansion ratio of 2, which is better (but not so far) than our 1.087 value. Yet this is achieved with a more complex code, that uses several cascaded bipartite graphs and a final Cauchy codec that protects the last level. Our LDGM codes are in comparison rather simple.
Since no public implementation of either the codes used in [88] or Tornado Z is available, only a limited comparison is possible. Therefore we can not compare such critical aspects as the encoding/decoding speeds, the maximum memory requirements, and the operational complexity (e.g. from [88] it is not sure that the same degree distribution always achieves the best results when varying the object size and FEC expansion ratio, which could create practical problems). If our results are lower than those achieved with the above patented codes, (1) they are not ridiculous though, and (2) the overall codec complexity is significantly lower.

10.3.3 Comparison with Rate-Less Codes

Fixed rate large block codes are known to have limitations, in particular when compared to rate-less codes[62]. This is essentially caused by the necessity to create and store on both ends (coder and decoder) the same parity check matrix. More specifically:

- *n is predefined:* The source must define beforehand the maximum number of parity packets that can be generated. There is no way to increase this number afterward, except by generating a new parity check matrix H_2 and starting all over again. But the global efficiency in that case remains limited since H and H_2 are totally unrelated. On the opposite a rate-less code avoids this problem and produces dynamically new parity packets as the need arises.

  This is in our opinion the main limitation of the LDGM Staircase/Triangle codes for some applications, but not all of them (e.g. this is probably not an issue for some distributed storage schemes where the number of replica is decided beforehand).

- *Parity check matrix storage requirements:* the LDGM Staircase/Triangle codes require that both ends store the same H matrix of size $(n-k) \times n$. On the opposite rate-less codes make no use of a parity check matrix. For instance, with the Online code, the packet index itself is a seed that defines the set of source packets it is the sum (see chapter 2).

  Yet our experiments have shown that the amount of memory required to store the H matrix with the LDGM Staircase code (and to a lesser extent the LDGM Triangle code) is relatively low compared the amount of memory required to store source packets, even if it regularly increases with the FEC ratio. This small size is made possible by the sparse property of H which enables the use of efficient data structures (Chapter 8). We therefore do not consider this point to be a practical issue with small to medium FEC expansion ratios, that is to say the situations where LDGM Staircase/Triangle codes perform the best.

- *parity packet storage at the encoder:* LDPC and LDGM Triangle codes require that the encoder stores all the parity packets produced since they may be used for producing further parity packets. Yet no such requirement exists with LDGM Staircase (except for the previously produced parity packet), which is an advantage when producing parity packets on-the-fly is sufficient. This situation is in that case similar to that of rate-less codes where parity packets are exclusively the sum of source packets.

- *n, or equivalently the FEC expansion ratio, are limited by practical considerations:* the parity packet storage requirements (and to a lesser extent the H matrix storage requirements) limit the the FEC expansion ratio. Besides, the inefficiency ratio also increases when the FEC ratio increases. There is therefore a incentive to use non over-estimated FEC expansion ratios.

In spite of these limitations, LDGM Staircase/Triangle codes are good codes with many potential applications. These codes are highly recommended when there is no need to produce high numbers of parity packets, while working on large objects (as in distributed storage applications, section 2.1.2). LDPC codes (in a broad sense, which includes LDGM codes) have also recently been adopted in the DVB-S2 standard, as a replacement of RSE, which is a sign of their potential. Otherwise, in environments where the FEC expansion ratio needs to be high, rate-less codes are clearly preferable. Such codes will be considered in future works.

10.4 Conclusions

In this chapter we provide an extensive performance analysis of four large block FEC codes, derived from LDPC, under various situations, and compare them to a widely used Reed-Solomon small block FEC codec.
We try to provide an exhaustive survey of the performances of these codes to fully understand their benefits and limitations, in order to use them in the most appropriate way.
An interesting result is the fact that a regular LDPC code has no interest at all, since it provides no erasure protection benefit compared to the LDGM Staircase/Triangle codes while having prohibitive encoding times (even if our codec could be improved from this point of view) and memory requirements.
Our results, in terms of decoding inefficiency, are not ridiculous when compared to previously published large block FEC performances, using patented codes, even if they remain lower.
Our conclusions is that the LDGM Staircase/Triangle codes have a high potential and can be advantageously used in many situations. Future works will extend the comparison with rate-less codes (e.g. the patent-free Online code [62]) and on new MDS codes like [40] that are capable to work on block sizes significantly higher than with traditional MDS codes.

Chapter 11

Analysis of FEC Codes for Partially Reliable Media Broadcasting Schemes

Contents

11.1 Introduction

This chapter analyzes a FEC-based partial reliability service in the context of a media (i.e. audio and video) delivery system like DVB or DAB (Digital Video/Audio Broadcasting) or based on the reliable multicast approaches seen in the context of this work (i.e. especially FLUTE/ALC).
As we have seen before, in order to be efficient, large scale content delivery protocols largely rely on the use of a Forward Error Correction scheme. However, the FEC decoding process may not end successfully if not enough packets have been received by the receiver (i.e. less than k or a little bit more). This is the case when reception is interrupted (e.g. a carousel-based transmission may be active only for a limited duration as it is the case in *push* mode) or is affected by a too important amount of losses (e.g. if a vehicle enters an environment with many obstacles that only enables an erratic connectivity). A partial reliability service should maximize the amount of original content reconstructed at a receiver, even if FEC decoding did not finish. The target of this work is therefore to *analyze the inherent capabilities of each FEC code to maximize this partial decoding, and to find the best operational conditions to achieve this goal.*
The work of the present chapter is also presented in [72].

The chapter is structured as follows: Section 11.2 presents the issues with a partial reliability service for RSE, LDGM Staircase and LDGM Triangle. Sections 11.3 and 11.4 introduce experimental results in specific scenarios. Finally we conclude this chapter.

11.2 Partial Reliability: the Solution Space

We now explain the various possibilities when designing a partial reliability service for a media delivery system.

11.2.1 Self Sufficiency versus Interdependency

A video content is composed of I, P and B video frames, and an audio content is composed of samples also called frames. All audio frames and the I video frames are self-sufficient entities, and the transmission scheme must absolutely preserve this self sufficiency property[1].
But P video frames depend on the previous I and P frames, and B video frames depend on the previous/next I and P frames. So in a partially reliable transmission scheme, there is an incentive to reconstruct the self sufficient frames first, before reconstructing the entire content. It leads to the idea of Unequal Error Protection (UEP) that will be considered in future works.

11.2.2 Object Definition

We call *object* an entity defined by the application and submitted to the transport layer as a whole. This exactly corresponds to the terminology used within ALC [45]. This definition is rather generic and therefore objects may largely differ, depending on the application: it can be an entire file, an audio track, or just a frame. If a large block LDGM-* FEC codec is used in the transport layer, the entire object, even if large, will probably be encoded as a whole (single block). But with an RSE codec, a large object has to be segmented into several blocks. As we'll see later on, this major difference will largely influence the efficiency of the partial reliability service.

11.2.3 Frame to Object Mapping

Several solutions exist to apply the notion of object to a media content, and control how FEC encoding is performed:

1. **Map each frame to a separate object and encode them independently:** the application can exploit a frame as soon as the corresponding object has been decoded (and all the other frames it may depend on with P and B video frames). Even if the frame size varies, depending on the bit rate and frame type, this size remains small, from several hundreds to a few thousands of bytes. Therefore there is no object segmentation, even with RSE. But LDGM-* codes have poor performance when operating on small block sizes, unlike RSE which benefits from its MDS feature [97]. So we can expect bad results with LDGM codes here. Additionally, since there is a high number of objects, all codes will suffer from the Coupon Collector Problem. So we can expect all codes to have bad performance.

2. **Map the entire content to a single "meta object" and encode it:** encoding is now performed over the whole meta-object for LDGM-*, and over blocks of maximum size for RSE. So the Coupon Collector Problem is totally eliminated with LDGM-* codes, and largely reduced with RSE (there are fewer blocks). Besides we know that LDGM-* codes perform very well with large objects [97], so their use seems rather promising.

   Since a receiver must be able to reconstruct individual frames of a partially received meta-object, the underlying structure of the meta-object must be communicated. This can be achieved within the transport protocol by means of a dedicated signaling header extension (e.g. we added a private ALC header extension that contains a description of the meta-object and that is sent periodically).

[1] This is the same requirement as when defining an RTP framing scheme with some multimedia content.

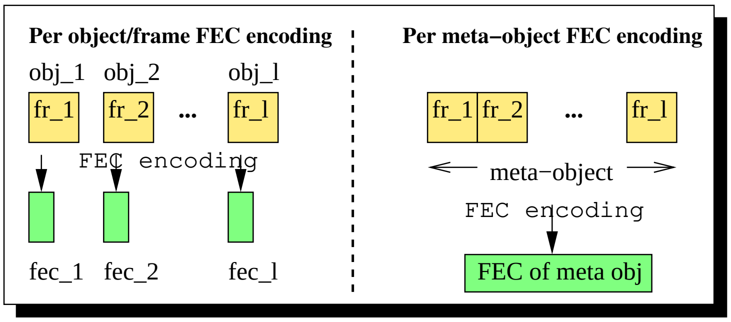


Figure 11.1: Per object versus per meta-object FEC encoding.

Finally, operating on meta-objects means that FEC encoding is done over several seconds or minutes of the media content, or even over the entire media file. A significant latency is thus introduced in the transmission model, which is an issue in case of real-time streaming, but is not a problem with the test cases considered in this chapter, or in our SVSoA proposal (see chapter 14).

11.2.4 Benefits of an Iterative Decoding System: An intuition

Intuitively, an iterative decoding algorithm should have a major advantage over MDS codes in partially reliable sessions. The reason is that decoding is done progressively and can be stopped at any time. A receiver can therefore exploit the subset of source packets received *plus* the subset of source packets already decoded. On the opposite, decoding with RSE is only possible once exactly k distinct packets have been received. If the session stops before this threshold, then no decoding is possible, and only the subset of source packets in the packets received can be exploited.
This fundamental difference is intrinsic to the FEC codec. However the intuitive conclusion that LDGM codes are more suited to a partial reliability service is not always true, as we will show later on.

11.3 One-to-One Frame to Object Mapping

11.3.1 Experimental Conditions

All the experiments use our LDPC/LDGM codec [77] (chapter 8) and a popular RSE codec [91]. We designed a basic application on top of these two codecs, derived from the `perf_tool` of our FEC codec. Given one or more objects, the application first performs FEC encoding: to k source packets of a block, the codec produces k parity packets, all packets being 512 bytes long. Then all the source and parity packets (of all blocks) are transmitted in a fully random manner, indefinitely, by choosing one packet out of all the possibilities (to mimic a random carousel transmission)[2]. The receiving application waits until it has received enough packets to decode the block (or all blocks with RSE) and stops.
As can be noticed, there is no explicit loss model or loss rate, because random transmissions are not affected by them (similar to the transmission schemes introduced in chapter 9). This approach simulates lossy transmissions with a random carousel like scheme, and is in line with transmissions like data broadcasting to cars [25], DVB IP Datacast [81], or a video streaming approach like [73].

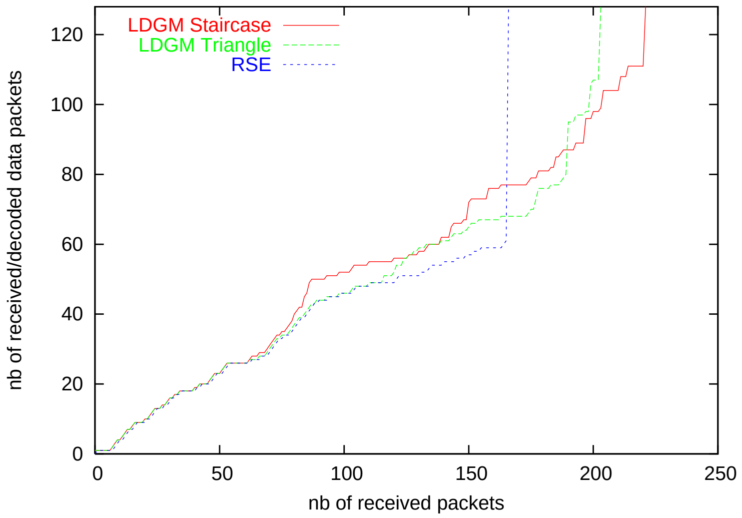
11.3.2 Performance With a Single Small Object

This first experiment illustrates the decoding behavior of each code. We consider a small object that fits into one RSE block (i.e. 128 source packets and 128 parity packets), and test all three codes using exactly the same packet reception sequence. Results are shown in figure 11.2 (a).

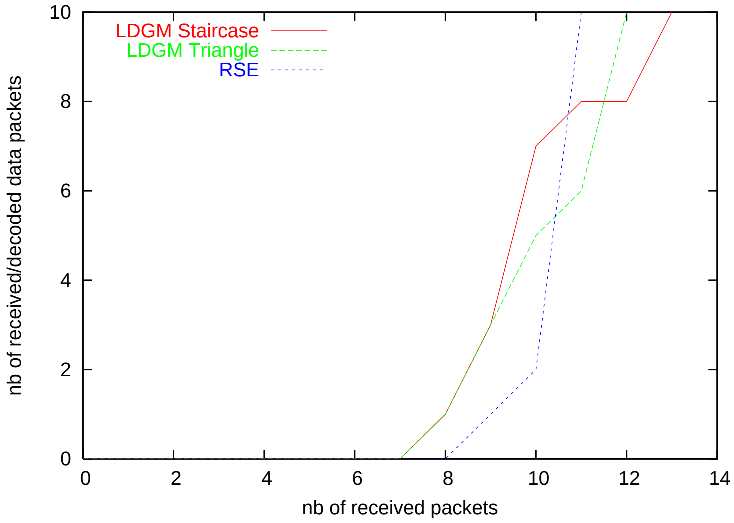
[2]Note that this is different from the transmission model used in chapter 9, since here we simulate a continuous (i.e. indefinitely long) carousel transmission.

Since RSE is an MDS code, it decodes the object first, after receiving 166 packets (this value is ≥ 128 because of duplicated packet reception made possible by the continuous random carousel transmission scheme). But RSE does not offer any partial reliability service in this scenario, and the source packets available earlier are those that have been received. The LDGM-* codes need more packets to decode the entire object. Yet source packets start to be decoded sooner, after 63 receptions for LDGM Staircase and 116 for LDGM Triangle. We also see that the Triangle version finishes the entire decoding sooner, but starts partial decoding after the Staircase version (because of extra relationships in the H matrix between all packets).

We did the same with a smaller object of size 10 packets, closer to an actual audio or video frame size. Figure 11.2 (b) shows the same general behavior as previously, at a smaller scale.



(a) Object composed of 128 packets



(b) Object composed of 10 packets

Figure 11.2: Number of decoded packets with a single small object.

11.3.3 Performance With a Large Number of Small Objects

Here we transmit 600 objects of size 10 packets each, for a total content of about 3 MBytes, or 6000 packets of size 512 bytes. Results are shown in figure 11.3.

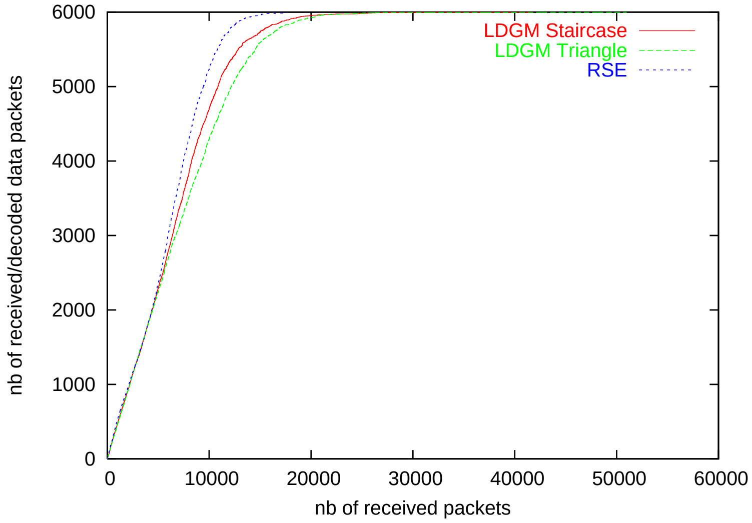


Figure 11.3: Number of decoded packets with a per object encoding and a large number of small objects.

As expected, the general performance behavior is rather bad with all codes, especially with LDGM codes: RSE needs 17651 packets, LDGM Staircase 41677, and LDGM Triangle 50971. These poor results, caused by the coupon collector problem and a sub optimal use of LDGM codes, are not acceptable.

11.4 Frame Aggregation in a Single Meta Object

11.4.1 Performance Without Inter-Frame Dependencies

Single Packet Frames:

We first analyze each code in case of a meta object of size 6000 packets, assuming each frame forms a single packet. Results are reported in figure 11.4. We see LDGM Triangle finishes decoding the first (after 9403 packets received), followed by LDGM Staircase (after 9792 packets received) and then RSE (after 10282 packets received).
But partial reliability requires that decoding starts as soon as possible. LDGM Staircase starts decoding some data packets the first, after 798 received packets, but it is clearly visible only after $\approx$ 3000 packets. At this point LDGM Staircase offers the best partial reliability service. RSE begins to decode the first block after 6634 received packets, and then quickly crosses LDGM Staircase after $\approx$ 7800 packets. At this point RSE offers the best partial reliability service. LDGM Triangle performs quite poorly since decoding is essentially done at the end. No partial reliability service is offered by this code, even if it exhibits the best decoding performance of all three codes!

Frames That Span Several Packets:

We now consider a more realistic situation where frames span several packets. We analyzed the number of decoded frames as a function of the number of received packets. To simplify, we assumed that all the frames have the same size. If the object size is 1 packet, then the previous results of figure 11.4 can be used. In figure 11.5 we reported the results for object sizes of 2,3,4 and 10 packets respectively. We clearly see that RSE offers in all cases the best partial reliability service. The RSE performance even becomes better compared to the LDGM codes as the frame size increases.

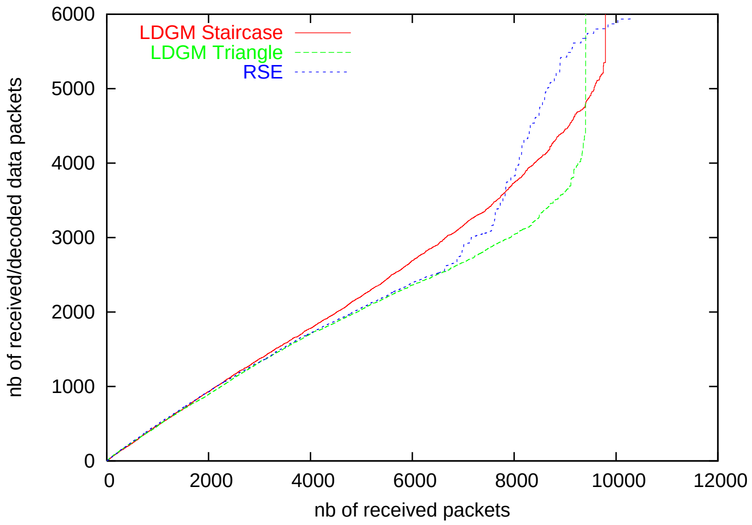


Figure 11.4: Number of decoded packets of a meta-object.

To better understand it, we also analyzed the time at which a given packet is decoded. Results are shown in figures 11.6. We clearly see that with RSE, packets are decoded per block, making a group of adjacent packets available to the application at the time of the decoding. On the opposite, both LDGM codes make the packets available in a completely flat and random manner. This is an issue if a frame spans several packets, since one of them may not be available. *Therefore RSE offers a better partial reliability service than LDGM codes when frames span several packets.*

11.4.2 Performance With Inter-Frame Dependencies

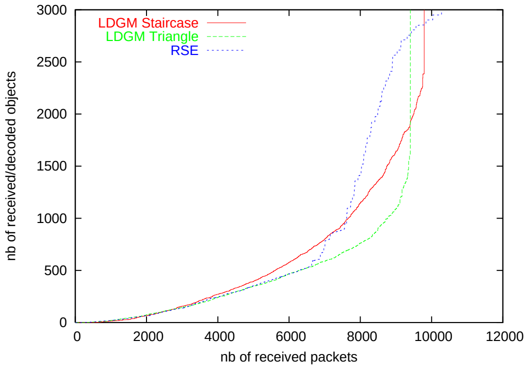
Finally we did similar tests but with an `IPPPPI` sequence of video frames, where `I` and `P` frames are respectively 10 and 5 packets long. Remind that a `P` frame depends on the availability of the previous `I` (and `P` if any) frame(s) of the current group of frames. The whole sequence is composed of 1000 frames, among which 200 are `I` frames. This test is therefore extremely close to an actual video media. Results, shown in figure 11.7, are rather close to that of figure 11.5. Here also *RSE is clearly the best solution to offer a partial reliability service.*

11.5 Conclusions

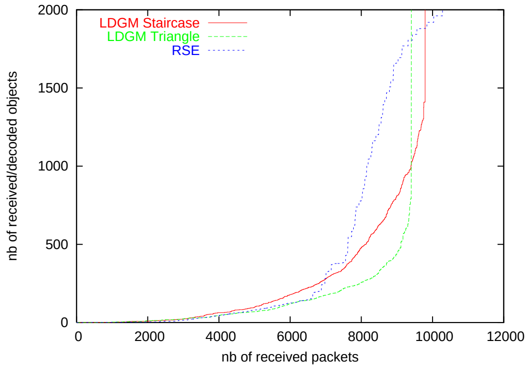
In this chapter we have analyzed the ability of three FEC codes – LDGM Triangle, LDGM Staircase and RSE – to offer a partial reliability service in case of media broadcasting schemes, that largely differ from more classical streaming approaches. We also introduced a meta-object encoding that largely improves the global efficiency of the broadcasting system. This analysis is quite innovative and we are not aware of any similar work. Traditionally only the global inefficiency ratio, which indicates how many packets must be received for decoding to complete, was considered. Our most important result is that LDGM-* codes, in spite of their iterative decoding approach and high efficiency, are not appropriate when partial reliability is desired. In contrast, RSE codes can more efficiently cope with this requirement. This is rather counter-intuitive and only detailed experimentations and analyzes led us to come to this conclusion.

Future work can further improve the scheme proposed. For instance it is a strong incentive to align frames with blocks, in order to avoid frames that straddle two blocks. A more detailed discussion and analysis on the impacts of the block size used is also a future research item.

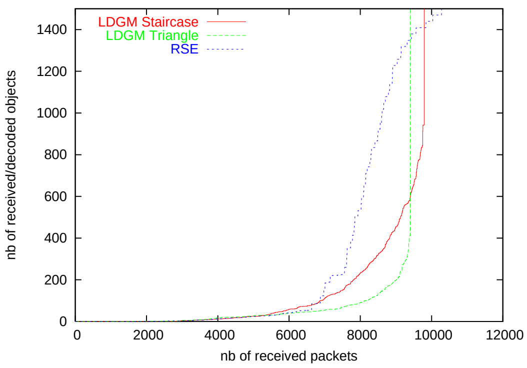
The proposed partial reliability scheme will be directly used in our Scalable Video Streaming over ALC (SVSoA) proposal [73] presented in chapter 14 and also conducted us to propose the file aggregation mechanism presented in chapter 13.



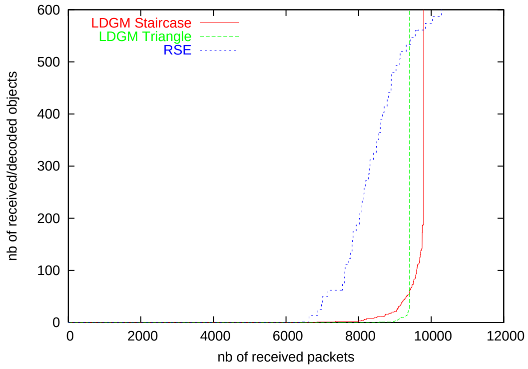
(a) Frame composed of 2 packets



(b) Frame composed of 3 packets

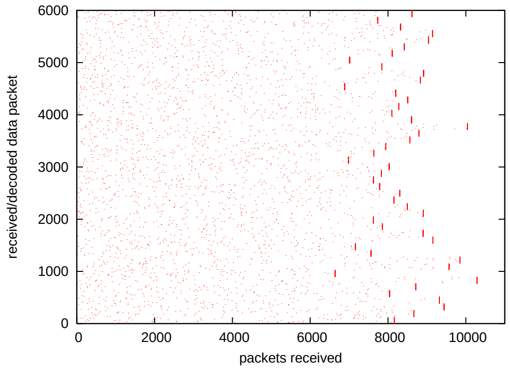


(c) Frame composed of 4 packets

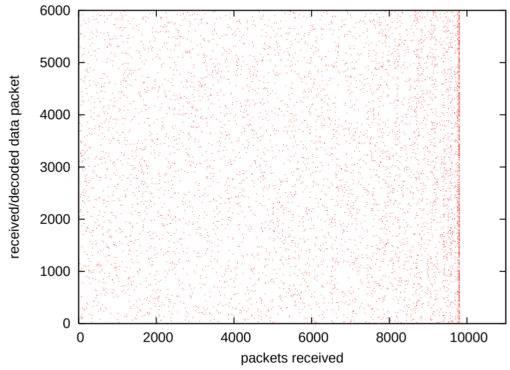


(d) Frame composed of 10 packets

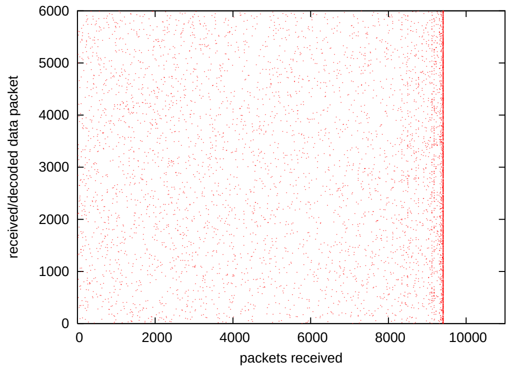
Figure 11.5: Number of decoded frames of a meta-object.



(a) RSE



(b) LDGM Staircase



(c) LDGM Triangle

Figure 11.6: Repartition of the received/decoded source packets of a meta-object as a function of time.

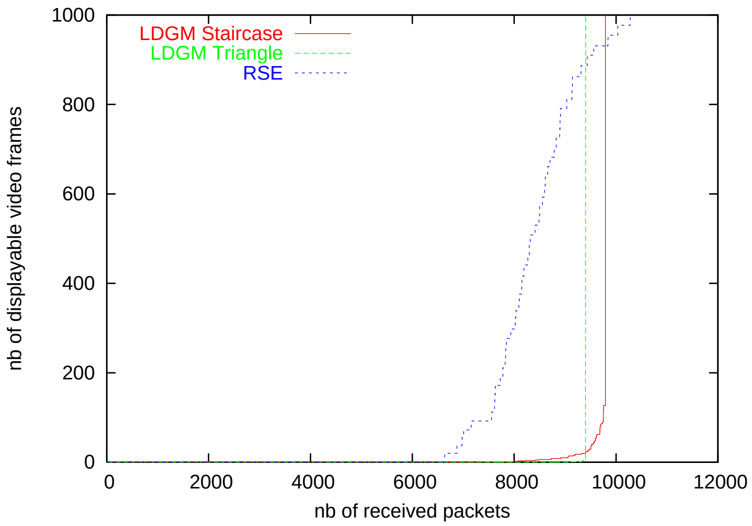


Figure 11.7: Number of displayable video frames of a meta-object with an IPPPPI sequence.

Chapter 12

Analysis of the Startup Behavior of Multilayered Multicast Congestion Control Protocols

Contents

12.1 Introduction and Related Works

12.1.1 Motivations

Using a reliable multicast content delivery system for the distribution of popular content to a large set of clients has shown to be very effective. The FLUTE file delivery application (RFC 3926[82]) and the underlying massively scalable reliable multicast protocol ALC (RFC 3450 [45]) are increasingly used to this purpose, and they are included in the technical specifications of the 3GPP MBMS service [4] and of the DVB-H IP Datacast service [26]. Thanks to them, a content can easily be delivered to several millions of clients, through a carousel: the content is continuously broadcast or multicast, and interested clients can join the session, download the content and leave the session whenever they want.

So far in this thesis, we focused on the FEC building block used by such approaches. As we have seen the performances are largely impacted by the FEC code used, and we evaluated the performances of several codes in this context.

A major question remains: how long does it take for a client to successfully download the content? Several parameters have to be taken into account: the content size, the available network bandwidth, the Forward Error Correction code features, and the packet loss model. When used in a *routed network* (like the Internet), a layered congestion control protocol is mandatory. In that case, another parameter is of importance: the congestion control protocol behavior during the startup phase, from the moment a client joins an ALC session and the moment an equilibrium is reached. Indeed, because of this congestion control protocol, on session startup, the client's reception rate progressively increases until it reaches a "fair share" of the available bandwidth between the source and the client (the exact fairness definition depends on the protocol used and is out of the scope of the present work). The time required to reach the steady rate is not so small as one would expect and has a major impact on reception performance. This is especially true when a client only downloads a small to medium size content, since the time required to reach the target rate will dominate the global download time.

The same considerations can be made for the SVSoA video streaming technique on top of ALC (see chapter 14), where we show the importance of the congestion control protocol used.

In this work we investigate the impacts of the startup behavior of three layered multicast congestion control protocols on reception performance. We first introduce a mathematical model and then analyze the amount of data actually received. Several experiments then confirm the models that we have elaborated. We assume in particular that receivers do not experience any loss, i.e. have not yet reached their target rate.

12.1.2 Layered Congestion Control Protocols

With ALC in a routed network, each receiver adapts dynamically how many layers to receive, depending on its access network and on competing traffic. This receiver-driven decision is taken by an associated TCP-friendly layered congestion control protocol (e.g. RLC [101], FLID-SL/DL [14] [59], or WEBRC [49] [48]). Choosing one of them can be dictated by the desire to have a standardized solution (use WEBRC then, since it is an RFC), or a simple, easy to implement protocol (use either RLC or FLID-SL then, even if they both have known limitations). However the client behavior will also largely differ according to this choice. For instance transmissions will take place either at some fixed predefined bit-rate on each layer (RLC, FLID-SL), or using a cyclic, dynamically changing bit-rate (FLID-DL, WEBRC).

The IGMP leave latency (delay between when the last receiver of a LAN leaves a multicast group and its effect) is of importance. This latency is usually 3 seconds (there are up to three instances of the IGMP polling message, with a typical 1 second polling delay each), but can be higher depending on the IGMP implementation. In addition to this delay, the multicast routing protocol itself can add its own pruning delay. The IGMP leave latency is known to affect the behavior of a statically layered protocols like RLC or FLID-SL. The only exceptions are the FLID-DL and WEBRC protocols that counteract this latency thanks to a dynamic layering approach. Note that some router constructors offer the possibility to completely remove the IGMP latency with IGMPv3, by enforcing the multicast router to remember who are the group members, and in such a case using RLC or FLID-SL raises no problem. Yet this remains an optional optimization, limited to IGMPv3 networks.

We now give a short overview of some multilayered congestion control protocols.

Receiver-Driven Layered Congestion Control (RLC)

In Receiver-Driven Layered Congestion Control (RLC) [101] transmissions take place at constant bit-rate on each layer. A receiver experiencing no loss can join higher layers at so called "increase signals" (indicated in the packet header). This mechanism helps to coordinate the behavior of receivers behind bottleneck links. The distribution of these signals and the transmission rate of each layer are specified in section 12.2.1. If the receiver experiences losses he drops the highest layer. Then a "deaf period" (of duration at least equal to the IGMP latency) is observed, during which a receiver ignores losses and increase signals. Its goal is to let the network prune the congested branch and then settle. After that the receiver can again drop layers if there are further losses or on the opposite add layers at the next increase signals if there is no loss anymore.

Fair Layered Increase/Decrease with Static Layering (FLID-SL)

Fair Layered Increase/Decrease with Static Layering (FLID-SL) [14], like RLC, relies on constant rate transmissions on each layer. The server also places signals into packets that indicate receivers when to join layers. However in FLID-SL these signal are based on a probabilistic function that tells whether to place or not an increase signal into a packet. The details of this function and the transmission rate of each layer are specified in section 12.2.2. Another difference is that the probing mechanism of RLC (used to test if joining a new layer is likely to be feasible or not) is no longer used in FLID-SL.

Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL)

Fair Layered Increase/Decrease with Dynamic Layering (FLID-DL) is a dynamic layered version of FLID-SL where the bit rate of each session layer is changing dynamically This is done in order to avoid the problems of the IGMP leave latency that static layered protocols like RLC or FLID-SL have. The idea is to continuously decrease transmission rate of each layer until reaching a point where no data is transmitted on this layer. After some time the layer restarts transmitting at maximum rate and again continuously decrease transmission rate. The receiver has to continuously join new layers in order to keep his reception rate constant. A receiver does not has to leave a layer in order to decrease reception rate, but just does nothing. Therefore the effect of the IGMP leave latency is bypassed.
However FLID-DL needs a prohibitive amount of overhead traffic, in particular because of the high number of IGMP messages. Therefore we focus on WEBRC, a protocol that also uses a dynamic layered approach (like FLID-DL) without its drawbacks.

Wave and Equation Based Rate Control (WEBRC)

Like FLID-DL, Wave and Equation Based Rate Control (WEBRC) [49] [48] uses the idea of dynamically changing transmission rates of each layer. In WEBRC the transmission rate of each layer evolves in waves. It is periodic, with an exponentially decreasing rate during the active period followed by a quiescent period. This feature reduces the number of layers needed (compared to FLID-DL), which also reduces the IGMP overhead, while solving the IGMP leave latency problem.
Unlike the congestion control protocols we have seen so far, that all rely on the losses observed by the receiver, WEBRC is equation-based. The available bandwidth is calculated by each receiver using an equation that imitates TCP behavior, and each receiver then adapts its reception rate accordingly.
The work of the present chapter is also presented in [74].
The remainder of this chapter is organized as follows: In section 12.2 we give the mathematical model of the startup phase of each congestion control protocol. Section 12.3 gives an account of some experimental results. In section 12.4 we explain the impacts and how to use our results in practice. Finally section 12.5 concludes this chapter.

12.2 Analysis of the Startup Phase

12.2.1 Startup Behavior with RLC

In RLC [101] a receiver experiencing no loss can add a new layer upon the reception of a dedicated "increase signal". These signals are exponentially distributed over the layers, using a factor of two, making the opportunities of adding higher layers less frequent than with lower layers. The minimum delay, t_l, after which layer l can be added, if no loss occurs, is: $t_l = (2^l - 1)t_0$, where t_0 is a fixed period (we use $t_0 = 0.25$ seconds in our experiments). Then each "increase signal" of layer l is repeated with a period: $T_l = 2^l t_0$. With RLC, the transmission rate of each layer is twice the rate of the previous layer. Therefore, when dropping a layer after a packet loss, this scheme performs a TCP-like multiplicative decrease. But adding a layer also doubles the reception rate. Since the increase signals are exponentially distributed over the layers, this is not compliant with the TCP slow-start

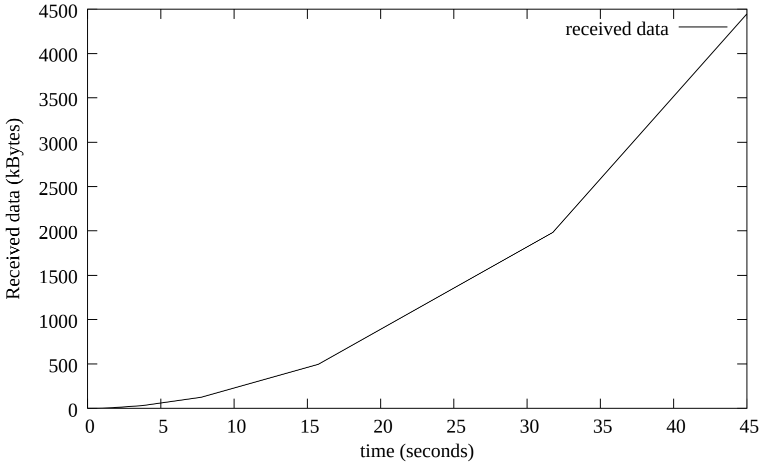


Figure 12.1: Amount of data received in the startup phase of RLC.

algorithm which follows an initial exponential behavior (TCP doubles the transmission rate each RTT in the initial startup phase).
The transmission rate of layer $l \in \{0; alay\_nb - 1\}$, where $alay\_nb$ is the total number of layers in an ALC session, follows a doubling scheme:

$$b_l = \begin{cases} b_0 & \text{if } l = 0 \\ 2^{l-1} b_0 & \text{if } l \geq 1 \end{cases}$$

Therefore, the amount of data received through a single ALC session in the startup phase, at time $t = i * t_0$, multiple of the RLC's time slot period, is:

$$\begin{aligned} & Rx(t = i * t_0) = \\ & data\_received\_on\_base\_layer \\ & + \sum_{l \in \{active\_upper\_layers\}} data\_received\_on\_layer\ l \\ = \; & b_0 \sum_{k=0}^{i-1} t_0 + \sum_{l:\, l>0 \text{ and } 2^l < i+1} 2^{l-1} b_0 \sum_{k=2^l-1}^{i-1} t_0 \\ = \; & b_0 t_0 \left(i + \sum_{0<l<\log_2(i+1)} 2^{l-1}(i+1-2^l) \right) \end{aligned}$$

If we only consider the moments when a new layer is added, i.e. if $\exists\ L : i+1 = 2^L$, then this equation can be simplified:

$$Rx(t = i * t_0) = \frac{i(i+2)b_0 t_0}{3} \tag{12.1}$$

Figure 12.1 shows the $Rx(i)$ curve as a function of time, when $b_0 = 11.9$ kbps and $t_0 = 0.25$ sec.

12.2.2 Startup Behavior with FLID-SL

We now consider the FLID-SL (Static Layer) congestion control protocol [14] which shares many similarities with RLC. The b_l can follow a multiplicative scheme:

$$b_l = \begin{cases} b_0 & \text{if } l = 0 \\ (C^l - C^{l-1}) b_0 & \text{if } l \geq 1 \end{cases}$$

where $C > 1$ is the multiplicative factor. [14] and [59] recommend to use $C = 1.3$.

The main difference between FLID-SL and RLC concerns the period T_l between two "increase signals" on layer l. T_l depends on a probabilistic function p_l which indicates the probability to increase the subscription layer in each time slot. On average T_l is given by:

$$T_l = \frac{TSD}{p_l}$$

where TSD is the Time Slot Duration. [14] proposes a heuristic for p_l to mimic TCP. [59] suggests a simpler scheme that we consider here:

$$p_l = min\left(1.0, \frac{20 * pkt\_sz * TSD}{C^l * b_0}\right)$$

We have to consider the two parts of this equation: for small l where $\frac{20*pkt\_sz*TSD}{C^l b_0} > 1.0$, and for l where $\frac{20*pkt\_sz*TSD}{C^l b_0} \leq 1.0$.
Let l_{min} be such that: $\frac{20*pkt\_sz*TSD}{C^{l_{min}} b_0} = 1.0$.

$$T_l = \begin{cases} TSD & \text{for all } l \leq l_{min} \\ \frac{TSD}{p_l} = C^l \frac{b_0}{20*pkt\_sz} & \text{for all } l > l_{min} \end{cases}$$

We can now divide the formula for the amount of received data into two parts.
For all layers l where $l \leq l_{min}$:

$$Rx_1(t) = t * b_0 + \sum_{l=0}^{l_{min}} ((t - l * TSD) * (C^l - C^{l-1}) * b_0)$$

For all layers $l = x + 1$ where $l > l_{min}$:

$$Rx_2(t) = \sum_{x=l_{min}}^{l_{max}} ((t - l_{min} * TSD - \sum_{y=l_{min}}^{x} (T_y)) * (C^{x+1} - C^x) * b_0)$$

with l_{max} such that: $t - l_{min} * TSD - \sum_{x=l_{min}}^{l_{max}} (T_x) = 0$.
The overall amount of received data is then:

$$Rx(t) = Rx_1(t) + Rx_2(t) \tag{12.2}$$

Figure 12.2 shows the $Rx(t)$ curve as a function of time, when $C = 1.3$, $b_0 = 11.90$ kbps, and $pkt\_sz = 576$ bytes. We use rounded values in our calculations for l_{min} and l_{max}.
We see that FLID-SL and RLC give similar results, even if FLID-SL, with the default parameters suggested in the literature, leads to a slower reception rate progression. We do not consider the FLID-DL (Dynamic Layering) scheme [14] here, since WEBRC replaces FLID-DL favorably.

12.2.3 Startup Behavior with WEBRC

WEBRC behaves differently than RLC or FLID-SL and is capable of adding layers much faster during the startup phase. Indeed the reception speed is multiplied by a factor of $C = \frac{4}{3}$ each epoch (by default 0.5 second), creating an exponential increase (TCP does the same since the transmission rate is doubled each RTT during the first stage of the slow-start algorithm), and then stabilizes around a "reasonably " fair share of the available bandwidth, determined through a TCP throughput equation. More precisely, every *epoch*, the reception rate is increased by a factor C. The amount of data received through a single ALC session in the startup phase, at time $t = i * epoch$, multiple of the *epoch* time slot period, is:

$$Rx(t = i * epoch) = \sum_{l=0}^{i-1} C^l * b_0 * epoch$$

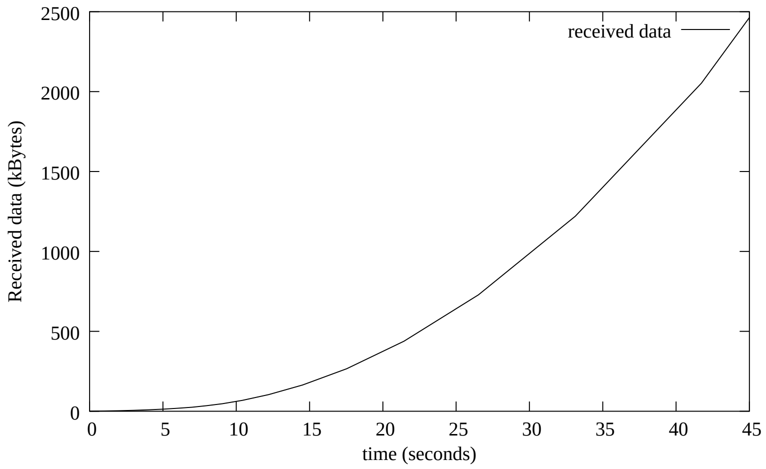


Figure 12.2: Amount of data received in the startup phase of FLID-SL.

By resolving the sum we obtain:

$$Rx(t = i * epoch) = \frac{(C^i - 1)b_0 * epoch}{C - 1} \tag{12.3}$$

This equation highlights the exponential increase of the reception rate (in $(\frac{4}{3})^i$ instead of i^2 with RLC).

12.3 Experimental Results

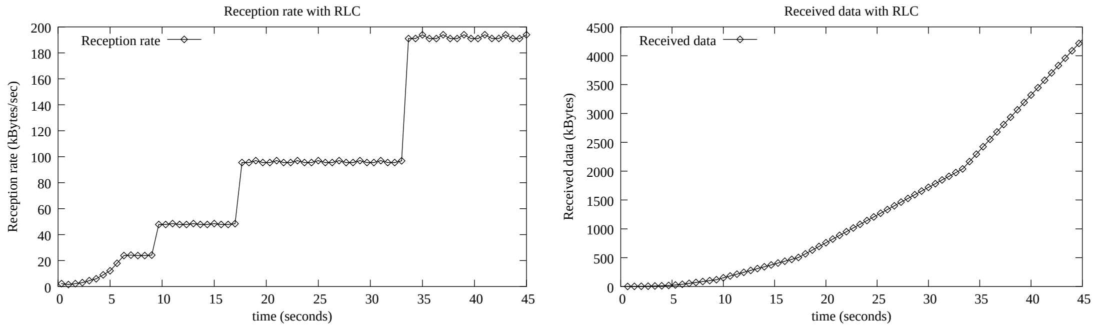
We have implemented RLC and FLID-SL within our MCLv3 library (see chapter 13) that implements the FLUTE/ALC family of protocols [93]. Since WEBRC is not implemented yet, experiments are restricted to RLC and FLID-SL.

12.3.1 RLC

The first experiments concern RLC. We use the same parameters as in section 12.2.1: $b_0 = 11.9$ kbps, and $t_0 = 0.25$ sec. Figure 12.3(a) shows the global reception rate at the receiver. We can clearly see the moments when new layers are added (e.g. $\approx 9sec$, $\approx 17sec$, $\approx 33sec$). This is slightly shifted to the right compared to the theory, where the values are $t_5 = 7.75sec$, $t_6 = 15.75sec$ and $t_7 = 31.75sec$ ($t_l = (2^l - 1)t_0$). This is mainly due to a small inefficiency in the timer's implementation, but it does not really impact the results.
If we compare figure 12.3(b) with the figure 12.1 we see that the amount of received data in practice is very close to the theory (e.g. at $45sec$: ≈ 4500 kBytes in theory and ≈ 4300 kBytes in practice). This small difference is here also caused by the slight shift to the right in the addition of layers that we mentioned before. Therefore, we can conclude that experiments have shown that the theoretical model matches the real implementation pretty well.

12.3.2 FLID-SL

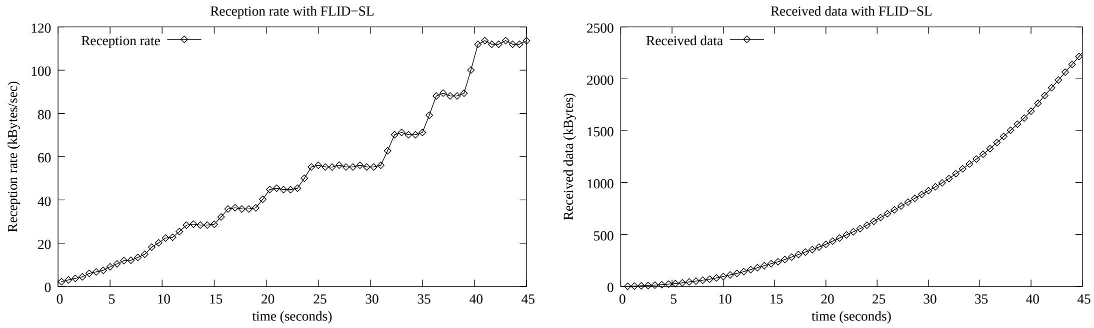
Concerning FLID-SL, we use the same parameters as in section 12.2.2: $C = 1.3$, $b_0 = 11.9$ kbps, and $TSD = 1$ sec. Figure 12.4(a) shows the global reception rate at the receiver. We see that a significantly higher number of layers are added than with RLC and offer a finer granularity of reception rates.
If we compare figure 12.4(b) with figure 12.2 we see here also that the amount of received data in practice is very close to the theory (e.g. at 45 sec: ≈ 2300 kBytes in theory and ≈ 2500 kBytes in practice). The difference can be explained by some inefficiencies in the implementation, in particular the fact that the sending rates on each layer are rounded and because of timer inferences. However we can here also conclude that experiments have shown that the theoretical matches the real implementation pretty well.



(a) Reception rate

(b) Amount of received data

Figure 12.3: RLC measurements.



(a) Reception rate

(b) Amount of received data

Figure 12.4: FLID-SL measurements.

12.4 How to Use these Results in Practice? Lessons Learned

12.4.1 Comparison When Ignoring the Target Rate Limitation

Our models allow us to calculate the total amount of data received at a given time when taking only into account the startup phase, before the target rate (assumed to be close to the TCP equivalent share of the bandwidth) is reached. Figure 12.5 compares this amount of received data using formulas 12.1 (RLC), 12.2 (FLID-SL) and 12.3 (WEBRC). It shows that WEBRC is the less impacted by the startup phase. A receiver benefits from the exponential behavior of the reception rate (and quickly reaches the TCP equivalent throughput). On the opposite, FLID-SL and RLC both experience a very slow reception rate increase. Therefore, *during small sessions, for instance when a client downloads a small file, RLC and FLID-SL download performance will essentially be dominated by the protocol startup behavior, unlike WEBRC.*

12.4.2 Comparison When Considering the Target Rate Limitation

Let's now assume that the target rate limits the reception rate to 10 Mbps, which is a reasonable value within a site network, or in new ADSL2 access networks, which are being more and more deployed. In that case, a receiver's download session behavior will be first controlled by the startup phase, and once the aggregate 10 Mbps threshold is reached (if ever this is the case!), download will then take place at this total rate [1]. It is important to consider this global reception bandwidth limitation

[1] We ignore here any target rate fluctuation or congestion control protocol limitation that may prevent a receiver to

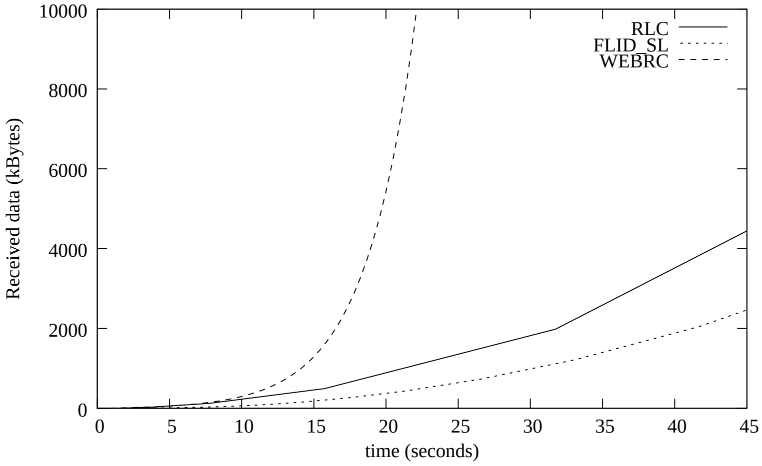


Figure 12.5: Amount of data received in the startup phase of WEBRC, RLC and FLID_SL. No bandwidth limitation.

since WEBRC in practice quickly reaches this threshold, which is not considered in section 12.4.1. The times when this target rate is achieved and the corresponding number of layers can easily be calculated using the b_l transmission rate formulas of section 12.2 first, and then the t_l formulas at which the layer is added.

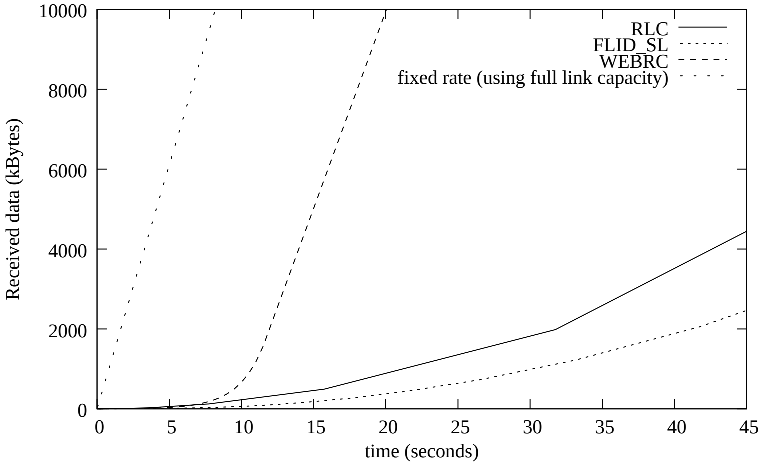


Figure 12.6: Amount of data received in the startup phase of WEBRC, RLC, and FLID_SL. Fixed 10 Mbps maximum bandwidth limitation.

This is shown in figure 12.6, where we use the same parameters as in section 12.2. The amount of received data with WEBRC, RLC and FLID-SL is also compared to a fixed rate 10 Mbps reception scheme, in order to better appreciate the global impact of the respective congestion control startup phases. WEBRC needs only $\approx$ 11.5 seconds to reach the target rate bandwidth, then the global reception rate remains constant. RLC requires $\approx$ 63.7 seconds, and FLID-SL $\approx$ 495.0 seconds (not shown in the figure). The number of layers needed to reach this bandwidth is respectively 9 layers with RLC and 25 layers with FLID-SL.

A Small Example:

Assume a client is downloading a 500 kByte file. The minimum reception time amounts to $\approx$ 15.75 seconds with RLC, $\approx$ 22.51 seconds for FLID-SL and $\approx$ 9.44 seconds for WEBRC. On the opposite, if no

keep this rate since the present chapter only focuses on the startup behavior.

congestion control is used (which is never recommended!), receiving at 10 Mbps, $\frac{500kBytes}{10Mbps} \approx 0.41$ seconds are sufficient to download the content! Figure 12.6 and this small example clearly show the *large performance gap between the different congestion control protocols and a fixed rate transmission*, and confirms the major performance differences between the various protocols.

Comparison with TCP:

[38] introduces formulas to model TCP with its slow start behavior. The download time can be calculated using the formulas on pages 279-283. With an RTT=200ms we obtain a download time of $\approx$ 2.29 seconds for the same 500kByte content. Therefore TCP is faster than all multilayered multicast congestion control protocols, which is not surprising since sender and receiver are closely synchronized.

12.4.3 Transmission Rate Granularity Considerations

From the above tests, one must not too quickly conclude that FLID-SL should definitely be banned. The RLC protocol reaches the target rate faster than FLID-SL because it uses a smaller number of layers, each of them having a higher transmission rate. This is efficient, but it also compromises the granularity with which a receiver can adapt to congestion situations. This is clearly visible when comparing figures 12.3(a) and 12.4(a). RLC reception rate granularity (factor 2) is far too coarse to enable an appropriate behavior compared to FLID-SL (factor 1.3 only), and adding a single layer with RLC can lead to a severe network congestion. This is why *FLID-SL is preferred over RLC in practice*.

12.4.4 Reception Inefficiency Factors

In practice ALC introduces several inefficiencies, because of:

- the packet scheduling scheme over the several transmission layers, since the same packet may be received on several layers. This issue may be eliminated if an expandable FEC code is used [58] (chapter 2) since an infinite number of fresh parity packets may be generated. Since in practice public implementations of those FEC codes not available (they are protected by many IPRs), the duplication problem remains and is all the more acute as the number of layers within the session is large;
- FEC inefficiencies, for instance caused by the coupon-collector problem, if a small block FEC codec is used, or by the intrinsic FEC inefficiency, if one of the various LDPC codes is used.

These factors must be considered if one desires to estimate the actual download time, since a few percents of packets in addition to the object size will be required. How large this additional time is, depends on the inefficiencies mentioned above. It is impossible to predict it in general since it depends on many configuration-specific parameters. An experimental evaluation is necessary for each use case, and one of them has been benchmarked in [95, 96]

12.5 Conclusions

We have compared the startup phase of three layered congestion control protocols, RLC, FLID-SL, and WEBRC, and we have introduced formulas that enable to calculate the amount of data received by a client at any time, before reaching the target reception rate. We used these formulas to determine the robustness offered by our SVSoA scalable video streaming scheme based on ALC (see chapter 14), and it was our first motivation for this work. In addition to that, the present chapter has shown that:

- the congestion control startup phase has major performance impacts *with RLC and FLID-SL, where this phenomenon dominates the reception rate during several tens of seconds*. This behavior is the result of the reactive approach used by these protocols (a client adds layers until it experiences packet losses that are interpreted as the sign of network congestion), and the rate at which layers can be added is deliberately limited in order to limit the possible congestion it may create when a client download rate approaches its fair share.

- reaching the target rate faster by using a smaller number of layers, each of them having a higher transmission rate, is efficient. This is the reason why RLC (that uses a 2.0 scaling factor) yields higher performances than FLID-SL (that uses a 1.3 scaling factor). But this solution also largely compromises the granularity with which a receiver can adapt to the equivalent TCP throughput, and *in practice FLID-SL is preferred over RLC*.

- *WEBRC is less impacted*, essentially because of its initial exponential rate increase. This is made possible by the equation-based approach of this protocol and a dedicated "slow start" mechanism (which is not so slow when compared to RLC/FLID-SL). Since a receiver cannot calculate a meaningful target rate from its measurements, it uses default values (which warrants the fast reception rate increase we noticed) and leaves this mode on some events (e.g. a packet loss, a sharp increase in the multicast RTT, or some inconsistency in the experienced reception rate).

Yet this work does not take position on the respective merits of the three congestion control protocols, and in particular on their ability to compete fairly with TCP and similar congestion-controlled sessions. In particular the fact that RLC and FLID-SL do not take any account the source/receiver RTTs is an intrinsic limitation.

Part III

Contibutions to Large Scale Content Distribution Applications

Chapter 13

FLUTE Extensions and Implementation

Contents

13.1 Introduction

Our application level contributions regarding FLUTE is described in this chapter. Our contributions are mainly standardization (by proposing the FLUTE file aggregation scheme) and implementation activities. No real experimentation nor benchmarking has been carried out, and this is left for future work. Our solutions are based on the research results obtained in the previous parts of this thesis, are used in the real world and have shown to work efficiently.
This chapter is structured as follows: Section 13.2 introduces a file aggregation mechanism for FLUTE. Section 13.3 describes our FLUTE implementation, which is part of our broader reliable multicast MCLv3 project. Finally section 13.4 concludes this chapter.

13.2 A File Aggregation Scheme for FLUTE

We introduce a logical and physical file aggregation scheme for FLUTE.
The logical file aggregation mechanism is a generalized grouping mechanism, allowing to logically group files.
The physical file aggregation scheme allows, additionally to a logical grouping, to more efficiently use FEC in the context of FLUTE, in particular when dealing with a large number of "small" files. Indeed, our previous results have shown that large collections of small files cannot be efficiently handled: (1) large block codes have bad performance on small objects; (2) a large number of objects leads to the coupon collector problem (see chapter 10).
Unlike a solution based on the creation of an archive, the object aggregation scheme (1) avoids the need to perform preliminary transformations on the content and (2) preserves the possibility to extract

a subset of the content (based on the considerations made in chapter 11), which may be critical aspect with some partially reliable broadcasting test cases.

13.2.1 Standardization Activities

The mechanisms presented in this chapter have been subject of standardization activities at the IETF. More precisely an Internet-Draft has been published [78], and the work has been presented at IETF meetings. The idea of physical and logical file aggregation has found good acceptance in the RMT IETF working group, and the mechanisms described here will hopefully be included in future standards.

13.2.2 Logical aggregation

The logical aggregation mechanism offers a means to logically group files without physically binding them to the same transport object. This functionality is desirable when transmitting a set of closely related files that will be used by the receiver in the conjunction with each other. The effect is to simplify the FLUTE-to-application messaging and processing overhead and to enable selective caching of files when it is not feasible to either promiscuously receive all files or explicitly indicate all wanted files in advance of joining the FLUTE session.
One example is an html page (file) with several embedded images. By labeling the web page file and all related image files as being part of the same group, the FLUTE receiver knows in advance the files he needs to download based on only the URI of the web page file. Without this grouping mechanism he would have to analyze the web page file and then deduce which other files are related and need to be downloaded.

The Generalized Grouping Mechanism

The generalized grouping mechanism is the basic mechanism that allows to to logically aggregate files. It allows each file of a FLUTE session to be labeled as being part of none, one or several logical groups. The grouping mechanism is achieved by adding the "Group" element to the FDT.
The "Group" element can be added to a "File" element, an "aggregatedFile" element (introduced in section 13.2.3) or to the "FDT- Instance" element. In the first two cases it specifies that the file (or aggregated file) is part of a group that is identified by the value of the element entry "Group". A "Group" entry at "FDT- Instance" level specifies that all files (and aggregated files) listed in the FDT Instance are part of that group.

Example

With this simple extension any type of relationship between files can be expressed. As an example we want to express the hierarchical relationship depicted in figure 13.1. A web site is composed of two html pages (file1.html and file2.html). file1.html contains the image file3.jpg, and file2.html contains the image file4.jpg.

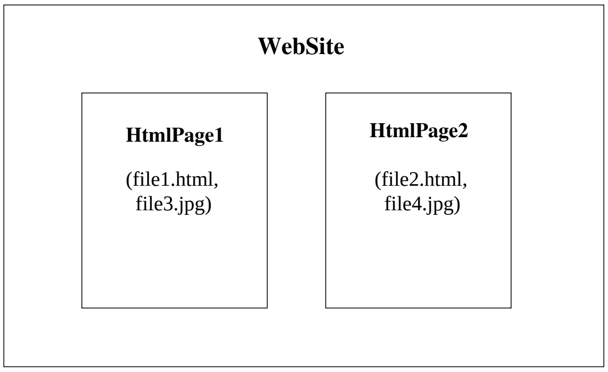


Figure 13.1: Example of a hierarchical relationship for a web site.

The hierarchical relationship can be expressed as follow: All files are part of the group "WebSite"; file1.html and file3.jpg are part of the group "HtmlPage1"; file2.html and file4.jpg are part of the group "HtmlPage2". This example is represented in the FDT Instance of figure 13.2.2. The FDT Instance describes the 4 files, plus a fifth one that is not part of the group "WebSite".

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:fl="http://www.example.com/flute"
xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
Expires="2890842807">
          <File
           Content-Location="file1.html"
           TOI="1"
           Content-Length="90"
           Content-Type="text/html">
               <Group>WebSite</Group>
               <Group>HtmlPage1</Group>
          </File>
          <File
           Content-Location="file2.html"
           TOI="2"
           Content-Length="100"
           Content-Type="text/html">
               <Group>WebSite</Group>
               <Group>HtmlPage2</Group>
          </File>
          <File
           Content-Location="file3.jpg"
           TOI="3"
           Content-Length="200">
               <Group>WebSite</Group>
               <Group>HtmlPage1</Group>
          </File>
          <File
           Content-Location="file4.jpg"
           TOI="4"
           Content-Length="210">
               <Group>WebSite</Group>
               <Group>HtmlPage2</Group>
          </File>
          <File
           Content-Location="file5.mp3"
           TOI="5"
           Content-Length="600"
           Content-Type="audio/mp3">
          </File>
</FDT-Instance>
```

Figure 13.2: FDT-Instance example.

Other file relationships can easily be expressed with the generalized grouping mechanism.

13.2.3 Physical aggregation

Motivations

The main idea of the physical file aggregation scheme is to aggregate a (possibly large) set of (possibly small) files into one large aggregated object, that is treated as a single transport object by ALC. The benefits of logical aggregation, described above, also apply to physical aggregation. However, a shared-fate model is introduced as the successful reception of one of the aggregated files is to some extent statistically correlated to the successful reception of one or more others. Thus, there is a strong incentive to only physically aggregate files that are logically related into the same aggregated transport object.

The physical file aggregation scheme is made possible by simple extension to FLUTE FDTs which provides a dedicated signaling mechanism, enabling extended FLUTE receivers to extract the files

within the large aggregated object.
With physical aggregation, FEC encoding is performed on a large object rather than on each individual file, which can be highly beneficial for transmission performance. Therefore this technique offers two specific transmission performance improvements:

1. the coupon collector problem [12], that is caused by the separate FEC encoding of each individual file when file aggregation is not used, is now significantly reduced or even totally eliminated. Now FEC encoding is done over a single object, whose size is perhaps inferior to the maximum block size permitted by the FEC instance used (this is especially true with Large Block FEC codes).

2. large block FEC codes perform better on large blocks than on small blocks, and using file aggregation offers more opportunities to use such codes whose performance is significantly higher than Reed Solomon codes [11].

The performance gain of these two aspects depends on several parameters such as the FEC instance used, the aggregated object size and the number of files. Detailed quantitative analysis and explanation of the impact of all these parameters are left for future work.
The specified physical object aggregation solution is significantly different from a solution that would create a single archive from the list of files (e.g. a gzip compressed tarball archive). With an archive the result is either the full reconstruction of all individual files (if enough packets have been received for decoding to complete at a receiver) or an hazardous result (since the archive will be corrupted, possibly damaging the archive headers which then prevents file extraction). Indeed, although FEC can counter the effects of packet erasures, if the number of packets received is too low for the FEC decoding process to finish, the received parity packets may turn out to be inconsequential. On the opposite, the specified physical object aggregation solution can offer a partial reliability service, i.e. it enables a receiver to reconstruct parts of the content even if the FEC decoding process has not finished. To that purpose, the physical file aggregation scheme can optionally preserve the possibility to decode and exploit a subset of the content, by informing the receivers of the size and position of individual files within the aggregated object.
Another motivation for having an object aggregation scheme compared to a basic archive based solution (e.g. tarball), is that no extra transformation (i.e. archive creation or extraction) is required at either the FLUTE sender or receiver. Everything is managed automatically by the transport mechanism according to transport- specific optimizations and can be transparent to upper applications (i.e. built on top of FLUTE), or enhanced by application hints on file relationships, without breaking the basic semantics of FLUTE sessions.

Multipart/mixed and Multipart/related MIME type object

An aggregated object is either a multipart/mixed MIME type object as defined in MIME Part two [30] or a mutlipart/related MIME type [41] if precise relationships between files have to be expressed (e.g. a dependency between a video file and its subtitle file). The aggregated object includes several files, each one delimited by the boundary delimiter defined in the MIME header. One body part (in MIME terminology) corresponds to one aggregated file.

Extending the FDT with Aggregated Object Information

The Aggregated Object Information (AOI) describes the aggregated object and its structure. The AOI is a logical view of all information needed to process an aggregated object. Its implementation within the FDT Instances as a "File" element for the aggregated object and a set of "aggregatedFile" elements, is presented later in this section.
The AOI must enable a receiver to:

- identify that an ALC object is an aggregated object,
- identify and have a description of the files being transmitted in an aggregated object,
- know the position (i.e. offset) and length of each file within the aggregated object.

Therefore the AOI must contain the following attributes:

- The Aggregated Object's TOI value
- The Aggregated Object's content type value, that must be set to "multipart/mixed" or "multipart/related".
- The URI of each file being aggregated
- The offset of each file within the aggregated object (not considering the boundary delimiter and the MIME header)
- The transfer length of each file within the aggregated object, or the content length if the file is not content encoded
- The number of files aggregated.

Therefore the new FDT-Instances are extended as follows: An aggregated object has a "File" element entry in the FDT, like normal files. A file being aggregated has a "aggregatedFile" element entry in the FDT. For this element the attribute "AOTOI", identifies the TOI of the corresponding aggregated object. The attributes "Content-Length" (or "Transfer-Length" if the file is content encoded) and "Content-Offset" are required "Content-Offset" specifies the offset of the file within the aggregated object.

Example

Figure 13.2.3 shows a simple FDT Instance using the physical file aggregation scheme. The files "http://www.example.com/menu/description.html" and "http://www.example.com/menu/details.html" are carried within the aggregated object, which has an TOI='1'.

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:fl="http://www.example.com/flute"
xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
Expires="2890842807">
          <File
           Content-Location="/AO1"
           TOI="1"
           Content-Length="830"
           Content-Type="multipart/mixed"
           Number-of-Files="2"/>
          <aggregatedFile
           Content-Location="http://www.example.com/menu/description.html"
           AOTOI="1"
           Content-Length="210"
           Content-Offset="100"
           Content-Type="text/html"/>
          <aggregatedFile
           Content-Location="http://www.example.com/menu/details.html"
           AOTOI="1"
           Transfer-Length="500"
           Content-Offset="320"
           Content-Encoding="gzip"
           Content-Type="text/html"/>
</FDT-Instance>
```

Figure 13.3: FDT-Instance example.

13.3 MCLv3/FLUTE implementation

We now present the FLUTE implementation elaborated during my thesis.

The MCLv3 project is an Open-Source GNU/GPL implementation of the two major reliable multicast protocols being standardized by the RMT IETF working group: ALC and NORM. The library also implements the file delivery protocol FLUTE. The project is composed of several C/C++ libraries and several applications built on top of it and provides an easy-to-use and integrated solution for reliable and highly scalable multicast delivery of data.
The project started in 1999 and continuously implemented the work done at the RMT IETF. The focus was made on the ALC approach, NORM being implemented only much later, and NORM is still in experimental state.
During my thesis, I contributed to parts of the ALC implementation, and fully implemented the FLUTE library and the FLUTE application.

13.3.1 General Architecture and Main Components

The MCLv3 library is implemented at user-level and is multi-platform. Systems currently supported are: Linux, Linux for PDAs, Solaris, FreeBSD and Windows.
An overview of the architecture of MCLv3 and its main components is given in figure 13.4. MCLv3 is logically composed of two parts: the NORM and the ALC part. Since NORM is not in the scope of this thesis we skip its description in the following.
MCLv3 includes the following libraries:

- *ALC library:* Accessed via the MCLv3 API, it almost fully implements the ALC protocol.
- *NORM library:* Accessed via the MCLv3 API, it partially implements the NORM protocol.
- *FLUTE library:* Accessed via the FLUTE API, it almost fully implements the FLUTE protocol.
- *TESLA library:* Based on A. Perrig's TESLA implementation [85], this library adds authentication to ALC as described in [27].
- *SinisterSDP library [22]:* Internally used by the FLUTE library to support the Session Description Protocol (SDP).

and uses the following external libraries:

- *XERCES-C library:* The FLUTE library strongly relies on xerces-c [29], the XML library of the Apache Software Foundation.
- *LDPC library:* MCLv3 uses the LDPC implementation [77] presented in chapter 8.

MCLv3 contains the following applications:

- *FCAST:* Built on top of the ALC and the NORM library, it implements a basic shell-based file transfer application.
- *FLUTE:* Built on top of the FLUTE library, it implements a basic shell-based FLUTE application. This is only an example and simplified FLUTE application, since it is expected that advanced uses of FLUTE will rely on the FLUTE API that enables external applications to exploit all the possibilities of the library.

FCAST

On top of NORM and ALC several applications and new libraries can be built. The MCLv3 distribution includes a simple file transfer application `fcast` and `fcastn`, based on ALC and NORM respectively. This file transfer application is different from FLUTE: The major difference between the two application come from the way file meta-data is carried within the ALC session: With FLUTE meta-data is carried as a separate ALC object. With `fcast/fcastn` meta-data is carried within the file object.
`fcast/fcastn` is not standardized anywhere (it derives from an old version of the ALC document [50]). However `fcast/fcastn`'s implementation is simpler (e.g. no need for XML parsing), and is only suited to situations where ALL clients are interested by ALL the objects sent within a given ALC session. It is therefore only kept in the distribution to demonstrate the basic capabilities of the NORM and ALC libraries.

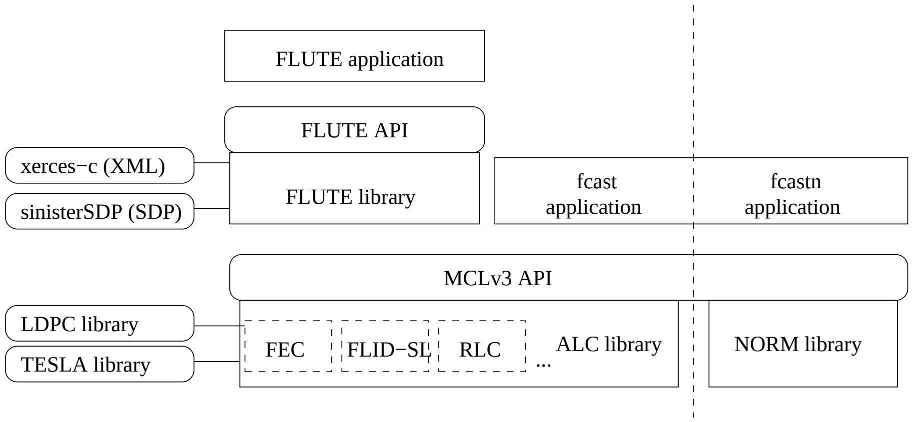


Figure 13.4: MCLv3 general architecture.

ALC

ALC can be accessed via the MCLv3 API, which is shared with the NORM library (i.e. it uses the same function calls).
The ALC implementation features:

- Conformance to RFC3450 and RFC3451 [45, 47]
- A congestion control building block, that supports RLC [101] and FLID-SL [59, 14].
- A FEC building block. This building block uses the LDPC implementation presented in chapter 8. It also includes a Reed-Solomon FEC implementation.
- IPv4 and IPv6 support.
- Source Specific Multicast.
- Object aggregation for improved transmission performance. This a concept close to the physical file aggregation mechanism [78] presented in section 13.2. The idea is also to aggregate several files into one transport object. The mechanism however does not use multipart/mixed MIME types and signaling is done within dedicated transport objects and not within the FDT (we aren't at FLUTE level anyway).

13.3.2 The FLUTE Component

FLUTE library

FLUTE has been implemented as a C++ library, built on top of the ALC library, with a dedicated easy to use FLUTE API. Writing an application on top of this FLUTE API enables a full and dynamic control of the FLUTE session, e.g. to dynamically add transmission cycles to the carousel, add/remove files to the carousel.
For XML parsing and DOM representations, the FLUTE library strongly relies on xerces-c [29], the XML library of the Apache Software Foundation.
The FLUTE implementation features:

- Conformance to RFC3926 [82].
- Fine carousel control: The transmission carousel at a FLUTE sender can be controlled in a detailed manner. Files can be added and removed during transmissions and new transmission cycles can dynamically be added or removed.
- FLUTE SDP Descriptors [65]: SDP can be used to describe the parameters required to begin, join, receive data from and end FLUTE sessions. Therefore the FLUTE library uses the SinisterSDP library [22]. The FLUTE SDP Descriptors draft [65] has been partially implemented.

A Quick Glance at the FLUTE API

The API is in C++ and is composed of the classes `Flute`, `FluteReceiver`, `FluteSender`, `FluteFileDeliveryCarousel` and `FluteFileInfo`.
For one FLUTE session an instance of the `Flute` class needs to be created. More precisely, `FluteReceiver` and `FluteSender` inherits from the class `Flute` and can been instantiated if a FLUTE receiver or FLUTE sender is needed. The API of the three classes are shown in figures 13.5, 13.6 and 13.7.

```
// destructor/constructor
Flute ();
~Flute ();
void abort ();

//get and set methods
... getXXX();
void setXXX (...);

// SDP methods
void createSdp (char* fileparam);
void parseSdp (char* fileparam);

// Are we are flute sender or receiver
bool isSender();
bool isReceiver();

// File Informations
class FluteFileInfo* getFileInfoList ();
class FluteFileInfo* getFileInfo (TOI_t toi);
```

Figure 13.5: API of class Flute

```
// destructor/constructor
FluteReceiver ();
~FluteReceiver ();

//get and set methods (receiver specific)
... getXXX();
void setXXX (...);

// file selection
void setSelectAll (bool on_off);
void selectTOI (TOI_t toi) ;
void unselectTOI (TOI_t toi) ;

// set callbacks
void setCallbackReceivedNewFileDescription (void*
 (*ReceivedNewFileDescription_callback) (class FluteFileInfo * fileInfo));
void setCallbackReceivedNewFile (void*
 (*ReceivedNewFile_callback) (class FluteFileInfo * fileInfo));
void setCallbackEndOfRx (void*
 (*EndOfRx_callback)(UINT64 BytesRecvd));

/ receiving method
UINT64 recv (bool blocking);
```

Figure 13.6: API of class FluteReceiver

The classes `Flute`, `FluteReceiver` and `FluteSender` defines several methods `setXXX` and `getXXX` that allows to set and get the sessions parameters, as for instance `getPort` and `setPort` for the port and `getTxRate` and `setTxRate` for the transmission rate. The functions `createSDP` and `parseSDP` can respectively generate and parse an SDP file that describes the FLUTE session.
At a receiver the `recv` function can be called in blocking or non-blocking mode. Callbacks notifies the application of new received files or file descriptors or the end of reception. The callbacks are set

```
// destructor/constructor
FluteSender ();
~FluteSender ();

//get and set methods (sender specific)
... getXXX();
void setXXX (...);

// the sender carousel
class FluteFileDeliveryCarousel *carousel;
```

Figure 13.7: API of class FluteSender

with the methods `setCallbackXXX`. A receiver selects and unselect the files he wants to receive using the methods `selSelectAll`, `selectTOi`, and `unselectTOI`.
At sender side we deal with carousels: one sender exactly knows one class `FluteFileDeliveryCarousel`. The API of class `FluteFileDeliveryCarousel` is shown in figure 13.8:

```
// destructor/constructor
FluteFileDeliveryCarousel (class FluteSender *flute);
~FluteFileDeliveryCarousel ();
void reset();

TOI_t addFile(const char* filename);
void removeFile(TOI_t toi);

UINT64 startTxCycles (INT32 nb_cycles, bool blocking);
void addTxCycles(INT32 nb_cycles);
UINT64 stopTx();

void setCallbackEndOfTxCycles(void* (*EndOfTxCycles_callback)(UINT64 BytesSent));
```

Figure 13.8: API of class FluteFileDeliveryCarousel

Files can be added to (via `addFile`) or removed from (via `removeFile`) the yet to be launched or ongoing transmission.
A carousel can be launched in blocking or non-blocking mode (method `startTxCycles`). In non-blocking mode the application can be notified by a callback (set with the `setCallbackEndOfTxCycles` method) at the end of transmission of a carousel. The carousel can be controlled in a quite fine grained manner: when starting a new transmission cycles we specify the number of transmission cycles (method `startTxCycles`); new transmission cycles can be added on the fly (method `addTxCycles`), or the ongoing carousel may be stopped (method `stopTx`).
The class `FluteFileInfo` contains all informations relative to a transmitted file. It contains methods `getXXX` to get file relative informations (e.g. `getContentLength` for the content length of the file). The file descriptors are delivered to the application, either using the callbacks (when a new file or file description was received), or by calling the methods `getFileInfo` and `getFileInfoList` of the class `Flute`.

FLUTE application

The Flute application is a small shell-based application built on top of the FLUTE API. Extending this simplified FLUTE application to add new functionalities is both feasible and straightforward.
Interoperability tests of MCLv3-FLUTE versus MAD-FLUTE [83] and Nokia's FLUTE have successfully been carried out in early 2004.

13.4 Conclusions

This chapter presented our contributions related to FLUTE.

We first introduced standardization activities regarding file aggregation mechanism in the context of FLUTE. The present work does not include any quantitative experiments or simulation with regards to the gain of the approach proposed, and this is left for future work.
Finally we presented the FLUTE implementation and the more broader reliable multicast MCLv3 library, which offers a complete suite of libraries and applications for reliable multicast content delivery. The library is continuously maintained and enhanced. It has become quite popular and is already being used in commercial and non-commercial products.

Chapter 14

Scalable Video Streaming over ALC (SVSoA)

Contents

14.1 Introduction

This chapter deals with the streaming of videos, that are either real-time or pre-encoded, to clients who receive and play information on-the-fly.
Designing a video streaming solution for an environment having no or a constrained back channel due for instance to scalability issues leads to many challenges.

We introduce SVSoA version 1 and version 2, two approaches that use a transmission technique close to the file broadcast methods presented so far in this thesis. We make no assumption on the target environment either, which can be either the Internet (if/when multicast routing is available), a site (e.g. a campus or an hotel), or a dedicated broadcasting network (e.g. a cable or DVB network). SVSoA can be used in all these situations.
Because of these assumptions, the client set is generally highly heterogeneous. This heterogeneity must therefore be considered to enable each client to receive the video stream that best fits with its networking and processing capabilities. [99] identifies three key aspects for the acceptance of video broadcasting: scalability, reliability, and quality of the reconstructed video. We fully agree with [99] and this paper emphasizes the reliability of transmissions in front of severe packet loss conditions and the stability of the video quality reconstructed.

14.1.1 Video Scalability

The advent of recent video codecs like MPEG-2, H.263+, MPEG-4, or H26L has largely improved the streaming possibilities, making it possible to optimize the video quality *over a given bit rate range* instead of *at a given bit rate*. The video scalability feature, also known as hierarchical video coding (both names will be used indifferently), refers to the possibility to see a video at several spatio-temporal resolutions by parsing appropriate portions of the bit-stream. In MPEG-2 and 4, three scalability techniques exist: *Temporal scalability*, *Spatial scalability* and *Qualitative (or SNR, Signal-to-Noise Ratio) scalability*, all three dividing the video in one base and one (rarely more) enhancement video layer. In all cases a partial reception of the enhancement layer will provide very little benefit [43]. On the contrary, with the Fine Granularity Scalability (FGS) [43] a partial reception of the (single) enhancement FGS layer provides an enhancement proportional to the number of bits decoded for each frame. The enhancement layer can therefore accommodate a wide range of bit-rates and offers the possibility to continuously adapt to the available networking bandwidth. But there is a cost and for the same transmission bandwidth, the quality is higher with non-scalable video coding. [42] discusses the Multiple Description (MD) video coding scheme that produces multiple independent layers of the video stream, each of the same importance. This property highly improves robustness, especially if the two layers follow different paths.
This discussion highlights several points: (1) video scalability is a complex feature that often produces a *single* enhancement layer; (2) splitting artificially this enhancement layer, or receiving only a subset of the associated data, does not always produce the expected result; (3) an exception is the MPEG-4 FGS scalability since the enhancement layer can be split into an arbitrary number of sub layers or can be partially received, at the cost of higher complexity. We will see that our proposal is compatible with any scalability approach and does not assume the presence of a fine grained hierarchical encoding.

14.1.2 Layered and Single Layer Streaming Approaches

Many approaches exist for video streaming [42]:

Layered Streaming Approaches

The streaming of scalable videos fits well with a transmission in cumulative quality layers [71]. A traditional solution consists in mapping these video layers onto several multicast groups. In order to perform congestion control, each receiver dynamically adapts the number of layers received according to the experienced losses [60]. To behave correctly, this solution requires a fine video layer granularity, and a temporal scalability scheme is almost always assumed. Another requirement is that packets sent on the base layer experience no losses, because such losses usually trigger an important distortion in the reconstructed video (inter-layer dependencies). One solution to provide this transmission discrepancy is to protect data sent on the base layer with FEC techniques [58]. Another solution is to rely on a QoS differentiation mechanism within the network (Int-Serv or Diff-Serv), and to affect packets of the base layer to a prioritized service [9] [102]. This is a major limitation since a QoS service must be deployed between the source and each potential client.
In [102] the authors introduce two Source-Adaptive Multi-layered Multicast (SAMM) algorithms that adjust the number of layers and their bit-rate depending on feedback information sent either by

network elements and/or by receivers. This approach has major practical limitations since both variants require special features for the routers (priority drop preference, flow isolation, and congestion notification with network-based SAMM).

Single-Layer Streaming Approaches

Another solution consists in having a single video stream, mapped onto a single multicast group [10, 71]. In that case the source adapts the transmission rate (e.g. by changing the video coding) according to RTCP feedback messages that give an indication on the experienced packet loss rate at receivers. Even if RTCP packets are rate-controlled (e.g. not to exceed 5% of the total session bit rate), this solution is not massively scalable. Besides, this solution is single-rate and consequently does not take into account the client heterogeneity.
A variant, called Simulcast in [43] and Destination Set Grouping (DSG) in [18], consists in generating multiple bitstreams of different bit-rates. Each client chooses the most adequate video bitstream according to its networking and processing capabilities. Switching to another bitstream dynamically is also possible. This solution addresses client heterogeneity but requires to decide, at coding time, for a fixed total bit rate, how many bitstreams should be generated and their bit-rate (DSG uses feedback messages for that).

Digital Fountain's Approach

Since this company proposes a streaming product that at first glance looks close to our proposal, we shortly discuss it here. Since no research report or paper has been published, this presentation is based on their web site [1]. The video stream is partitioned into blocks. The length of the blocks may vary depending on the application or the network features, yet it remains short, around 100 ms. Each block is protected with their proprietary FEC code, and the protection amount chosen depends on the network properties. Using an FEC code allows receivers to reconstruct a block if it received an amount of data equal in length to the original block. This approach is not designed for massive scalability, video scalability, nor receivers heterogeneity (and nothing is said about congestion control).
Our proposal completely departs from the layered or single layer approaches. It vaguely resembles to that of Digital Fountain, but neither the application area nor the techniques are the same as ours. In fact SVSoA largely relies on ALC.
ALC was originally designed for the massively scalable and reliable content delivery, and media streaming was never mentioned to be a possible application. Streaming is not the original intent of ALC. We however show in this paper that when used properly, ALC is suitable for media streaming.
The work of the present chapter is also presented in [70, 73, 69].
The remainder of this chapter is organized as follows: Section 14.2 introduces the general ideas of SVSoA version A. Section 14.3 explains how to initialize the various parameters and what are the associated trade-offs of this version. Section 12.3 introduces some experimental results obtained on a local testbed with a full implementation of our proposal and of the ALC/RLC protocols. We then introduce SVSoA version 2 in section 14.5, with its principles a robustness evaluation and some experimental results. Finally Section 14.6 concludes the paper.

14.2 SVSoA version 1

14.2.1 Principles

The SVSoA approach relies on ALC (and associated protocols)/UDP/IP as the transport/network layers, and is placed beneath RTP and the server or player application. The general architecture is illustrated in figure 14.1. Note that no RTCP back channel (e.g. to carry feedback information to the source) is used in SVSoA.

Sender behavior

Let's consider a server who needs to stream a video that has been hierarchically encoded using one of the scalability schemes of section 14.1.1. The video consists of a base layer plus one or more

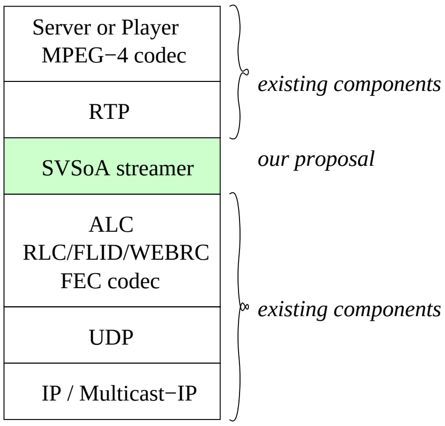


Figure 14.1: General architecture of the SVSoA approach.

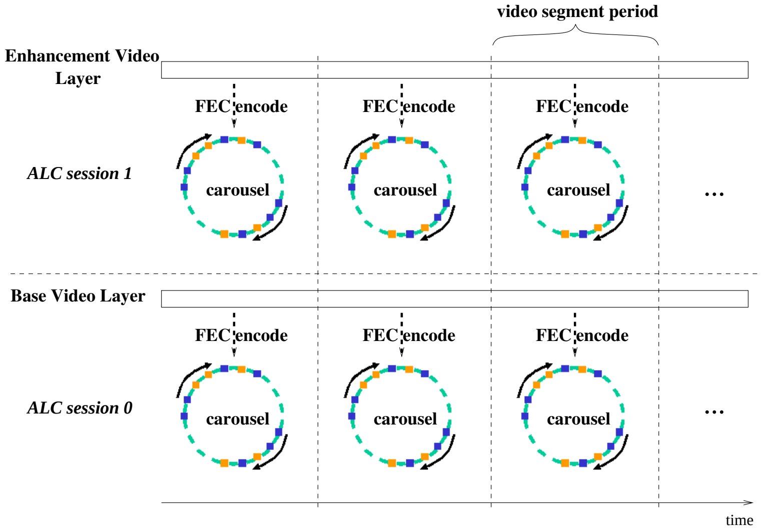


Figure 14.2: SVSoA version 1 sender behavior.

enhancement layers. Unlike many previous works, we do not assume the presence of fine granularity, so having a single enhancement layer is sufficient.
The sender first partitions the video stream into *segments* of approximately the same duration, VSD (Video Segment Duration). By default $VSD = 60$ seconds, but other values are possible (section 14.3). Each video layer produces a *block*, of duration VSD. Each block is then sent independently on a *distinct ALC session* and thus on a different set of multicast groups as shown in figure 14.2 (note that video layers and ALC layers are two different notions). After VSD seconds, the server automatically switches to the next segment (at $t0+VSD$ on figure 14.2), and for each ALC session, the transmission of block n is stopped and replaced by block $n+1$.
During each period, the packets of a block are not sent sequentially but in a random order and cyclically, in order to offer an "on-demand" delivery mode. FEC packets included by ALC in the data stream enable receivers to efficiently reconstruct some missing packets (either lost or not-yet received). This on-demand mode is required since receivers do not necessarily join at the beginning of a block transmission, especially on ALC sessions 1 and above. This is one of the reasons why ALC is absolutely required.

Receiver behavior

At the beginning of a period or when a new receiver joins the SVSoA session, a receiver first subscribes to the ALC session where the base video layer of the current segment, n, is sent (from t_0 to t_1 in

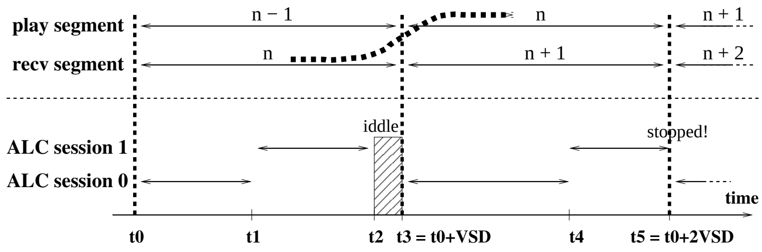


Figure 14.3: Receiver behavior.

figure 14.3). When this block is successfully received, the receiver subscribes to the ALC session of the next enhancement video layer (from t_1 to t_2). This process stops (1) when all blocks have been successfully received, if ever (e.g. at time t_2), or (2) when the transmission of the next segment, $n+1$, begins. When transmissions for segment $n+1$ start, the receiver plays the video of segment n and switches to the first ALC session, and so on. Therefore a receiver always plays the previous video segment while receiving the current one, which of course introduces a playing latency of VSD seconds (section 14.2.3).
In order to keep receivers synchronized a descriptor is sent by the source for each video block with the following information: the number of frames in the current block, the last RTP timestamp, and the duration, VSD, of the current segment. With this information a receiver knows when he completely received a block, and can detect the end of the current segment in order to switch to the next one. These descriptors are sent out-of-band on a dedicated control session.

Definition of ALC Objects According to the Video Scalability Scheme

The Application Level Framing (ALF) paradigm [19] tells us that each object should be autonomous and contain enough information to be processed by a receiver independently of other objects. We applied this principle and identified several framing possibilities: figure 14.4 (a) shows a possible framing in case of non-FGS spatial scalability. Since a receiver cannot take any benefit from receiving a subset of an enhancement frame, each frame is carried as a distinct ALC object. Figure 14.4 (b) shows the ALC object definition in case of a temporal scalability video. In this example each I and P frames of the base layer are independently carried in a different object, as well as each B frame of the enhancement layer (note that many other I/P/B mappings are possible).

14.2.2 Benefits of the SVSoA version 1 Approach

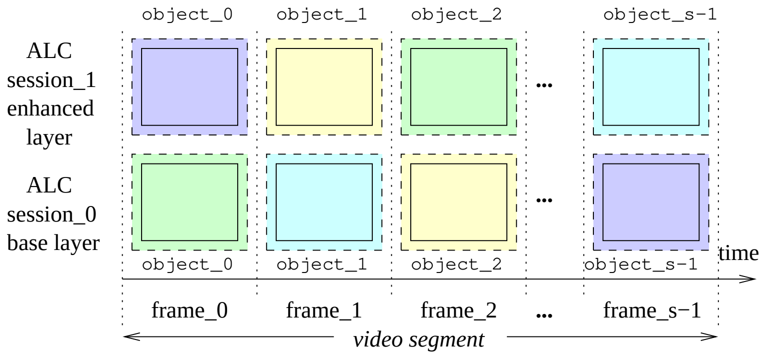
The benefits of this approach are numerous, many of them being derived from the use of ALC as the underlying transport protocol.

Massively Scalable

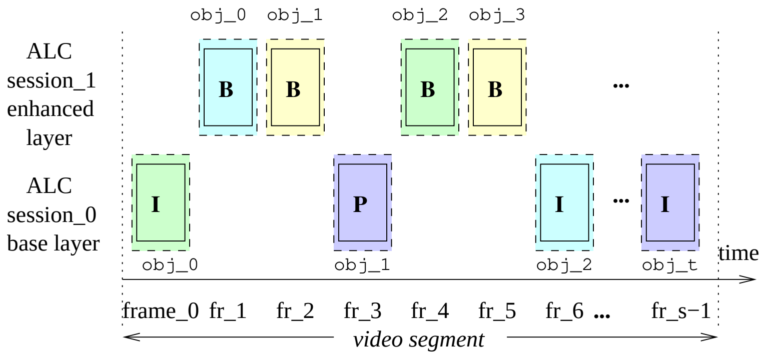
Because no feedback of any kind is used at either ALC, the layered congestion control protocol, or SVSoA, it makes no difference to the video server whether there are very few or several millions of simultaneous clients. Note that RTCP, the control protocol associated to RTP, is not used at all by SVSoA.

Exploits the Intra-ALC Congestion Control

Our approach takes advantage of the TCP-friendly layered congestion control protocol used within each ALC session. SVSoA automatically benefits from the latests developments in TCP-friendly congestion control protocols. This is a major asset since a recent protocol like WEBRC proved to be highly effective: receivers quickly reach the equilibrium point, achieve a good TCP-friendliness and are not affected by the IGMP leave latency issue [49]. The available bandwidth is always used optimally.



(a) Framing with Spatial Scalability



(b) Framing with Temporal Scalability (with an IBBPBBP sequence)

Figure 14.4: Object framing according to the video scalability scheme.

Immediately Deployable Anywhere

The SVSoA approach does not rely on any privileged transmission service nor on any specific feature within the backbone and can therefore be immediately deployed anywhere. This is made possible by the fact that a client receives only one video layer at a time, starting by the most important one (base layer). This solution therefore maximizes the probability of receiving the most important data correctly. On the opposite, traditional layered streaming approaches often rely on the presence of a QoS mechanism within the core backbone. This is in practice a major limitation which adds much complexity to the solution and restricts its use.

Addresses the Heterogeneity of Clients

Because ALC addresses the heterogeneity of clients, each SVSoA client receives the amount of video data made possible by its access network, independently of other clients. Besides, all clients are guaranteed to receive a minimum video quality before trying to receive any enhancement information.

Robust in Front of Packet Loss Bursts

Because ALC is a reliable protocol, packet loss bursts are easily recovered, even in case of long lasting bursts (e.g. several tens of seconds), without any brutal impact on the video quality perception. The only requirement is that enough time is left to enable a receiver to receive at least the base video layer during the segment duration. This is in line with [99] that shows that long bursts of packet losses (several seconds) occur quite often in large local area networks.

Independent from the Video Scalability Scheme

Another benefit of SVSoA is that the congestion control efficiency does not depend on the number of the enhancement layers provided by the scalable video codec, and the nature of scalability used by this codec. On the opposite, traditional approaches that rely on a direct mapping between video layers and transmission layers (multicast groups) assume the presence of a fine granularity video encoding. In practice this granularity is usually very low (e.g. the MPEG-4 ISO reference codec we used produces a single enhancement layer).

A Simple End-to-End Solution

Our proposal follows an end-to-end approach. It is a great asset compared to solutions like Content Delivery Networks (CDN) that can also be used for streaming. In that case, a complex distributed infrastructure must be designed, deployed and managed. Besides, video contents are replicated at several locations within the network. No such things are required with an end-to-end solution.

14.2.3 . . . And the Price to Pay

A High Playout Latency

When a new client joins an ongoing SVSoA session, this client experiences an *initial join latency*: $BLRT \leq initial\_join\_latency < VSD + BLRT < 2 * VSD$, where $BLRT$ (Base Layer Receive Time) is the minimum time required to get the whole base video layer. The minimum join latency is experienced when the client joins the session $BLRT + \epsilon$ seconds ($\epsilon \ll 1$) before the end of the current segment since he has enough time to get the whole base video layer and can display it immediately after switching to the following VSD period. There is no enhancement layer during this first VSD period but the most important information is displayed. The worst join latency is experienced when the client joins the session $BLRT - \epsilon$ seconds before the end of the current segment, since he needs to wait an additional VSD period.
The second source of latency is the *playing delay*: the video playout is always delayed by the segment duration parameter, VSD (typically 60 seconds, section 14.3.6). This feature prevents using SVSoA when interactivity (e.g. with tele-teaching) or immediate delivery (e.g. for a sport event coverage) are required.

Additional Traffic

Another drawback is a high cumulative transmission rate at the source, since all layers for all ALC sessions are active. Yet multicast routing limits the traffic carried on the backbone by avoiding transmissions on branches that do not lead to a receiver. In practice, if there is no receiver, the first hop multicast router prunes the traffic totally. Traffic is then restricted to the source's which is rarely an issue.
Let's now consider a client. ALC introduces several inefficiencies: (1) data and FEC packets are of finite number and duplications occur (e.g. the same packet can be received on two different layers at different times, or a packet for an already decoded block can be received later on); (2) "non-systematic" large block FEC codes like LDPC have intrinsic decoding inefficiencies, expressed by their inefficiency ratio. Some additional traffic will therefore be received, which is unavoidable with a reliable multicast protocol. In section 14.4.2 we show that even in a rate limited environment, SVSoA behaves efficiently and the extra traffic, in fact, enables clients to recover losses.

14.2.4 RTP/UDP versus RTP/ALC/UDP

The SVSoA approach relies on ALC/UDP/IP as the transport/network layers, and is placed beneath RTP and the server or player application. Classical streaming approaches like proposed in [4] only rely on RTP/UDP.
ALC offers a very flexible framework. As mentioned in Chapter 3, different types of building blocks can be plugged into ALC. It is therefore much simpler to add new features when new requirements comes

up (e.g. security), or if the used environment changes (e.g. adapt the congestion control protocol). RTP has no such framework and is therefore less flexible.
The flexibility issue also comes up when looking at the FEC mechanism used with RTP/UDP solutions. At this time there is no generic FEC building block for RTP. Some solutions like [4] were proposed and are discussed in the IETF AVT working group. But the proposed mechanism are less flexible (e.g. restricted to systematic FEC), and quite complicated.
ALC is designed to achieve good results where massively scalability is of importance. This is not the case of RTP, and some features have to be disabled to adapt it to this environment. Indeed by suppressing the RTCP feedback channel RTP can become massively scalable, but rate adaption (which is in fact the congestion control of RTP) is then no longer possible.
One drawback of ALC is, that it introduces very large packet headers. This overhead can be considered as non negligible compared to a streaming solution only based on RTP. This drawback can however be bypassed by introducing header compression protocols as RoHC [11] (RFC 3095). Even if RoHC does not support ALC at the moment, we believe that it can be easily extended to that purpose.

14.3 Analysis of the SVSoA version 1 Parameters

When deploying our solution some parameters must be adapted to the target environment (e.g. is it deployed in a closed environment like an hotel, or in the Internet), and to the video features (e.g. the bitrates of the base and enhancement video layers). In this section we explain how to optimally initialize two key parameters:

- the video segment duration (VSD), and
- the base transmission rate of each ALC session (b_0).

Several contradictory aspects must be considered when choosing a value for VSD. The first idea is to have a very short VSD in order to reduce:

- the storage capacity required at the source and at receivers, and
- the initial join latency and the playing delay.

But several considerations are against short video segment durations:

- the impacts of the IGMP leave latency when switching between two ALC sessions,
- the impacts of the congestion control protocol during the startup phase,
- and the maximum loss burst length that can be transparently recovered.

We now provide a theoretical analysis of each of these aspects.

14.3.1 Storage Requirements

The first limitation is the required storage capacity *at a server*. Since efficiency requires that packets are sent in a random order in each ALC session [95], the whole data set is usually stored in physical memory rather than on disk in order to avoid inefficient random I/Os. A direct consequence is that the available storage capacity (i.e. RAM) is quickly limited, and this is all the more true as the same server can stream several video contents simultaneously. The storage requirements amount to:

$$source\_storage\_req = VSD * cumul\_enc\_rate * FEC\_ratio \tag{14.1}$$

where $cumul\_enc\_rate$ is the cumulative video encoding bit rate over all layers.
Figure 14.5 illustrates the memory requirements at a source as a function of VSD for various encoding rates and
$FEC\_ratio = 2$ (as many FEC packets as original packets). It shows that a high quality video encoded at 2 Mbps requires only 29.3 MB of storage capacity in the ALC component with $VSD = 60$ seconds, which is fairly reasonable. Having higher VSD values (e.g. 120 seconds) is not a problem either since

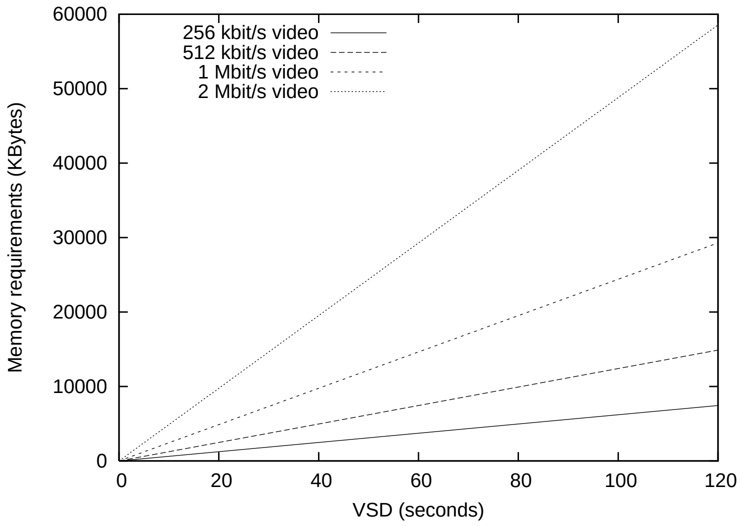


Figure 14.5: Storage requirements at a source.

the required memory only amounts to 58.6 MB. *We therefore consider that the storage capacity at a server is not a problem.*
The memory requirements *at a receiver* are lower since the FEC_ratio has no influence here (FEC decoding takes place immediately, as soon as enough packets have been received). But two segments can be buffered, the one being displayed and the one being received (in practice data displayed is immediately freed, so equation 14.2 is a pessimistic upper bound). Therefore:

$$receiver\_storage\_req = 2 * (VSD * cumul\_enc\_rate) \tag{14.2}$$

Besides, unlike a video server, a receiver is typically involved in a single video stream at a time. Buffering can yet be a problem for low end receivers (e.g. mobile devices). But in that case, because of the limited bandwidth of the access network and/or the limited display capabilities, the video encoding rate will also be low, thereby limiting the storage requirements (e.g. 1.9 MB with a 128 kbps video, and $VSD = 60$ seconds). *We therefore consider that the storage capacity at a receiver is not a problem.*

14.3.2 Impacts of the IGMP Leave Latency

The second limitation are the *impacts of the IGMP leave latency*, i.e. the delay between when the last receiver of a LAN leaves a multicast group and its effect. This latency is usually 3 seconds (there are up to three instances of the IGMP polling message, with a typical 1 second polling delay each), but can be higher depending on the IGMP implementation. In addition to this delay, the multicast routing protocol itself can add its own pruning delay. Let $igmp\_leave\_lat$ be the sum of these latencies (assumed constant).
It is well known that the IGMP leave latency will affect the behavior of a layered congestion control protocol like RLM, RLC or FLID-SL. The only exceptions are the FLID-DL and WEBRC protocols which counteract this latency thanks to a dynamic layering approach [14] [49]. Let's now assume that a dynamic approach is used.
Even with such a protocol, our approach is still affected by the IGMP leave latency whenever a receiver changes of ALC session, for instance to receive the enhancement layer, or when switching to a new video segment. During $igmp\_leave\_lat$ seconds, packets of the previous ALC session still flow up to the receiver LAN, thereby preventing a normal behavior of the new ALC session. Two cases must be considered:

- the host is the only client in the LAN: in that case during $igmp\_leave\_lat$ seconds, packets of the previous ALC session still flow up to the receiver LAN where no receiver exist any more, thereby preventing a normal behavior of the new ALC session.

- other clients in the LAN are still joined to the previous ALC session. In that case two (or more) ALC sessions will coexist for some time. This second case highlights a limitation of the host heterogeneity handling with layered multicast approaches: heterogeneity is managed with a LAN granularity, not a host granularity.

The impacts of this latency are given by equation 14.3:

$$igmp\_ineff\_ratio = \begin{cases} (vlay\_nb * igmp\_leave\_lat)/VSD & \text{if } vlay\_nb > 1 \\ 0 & \text{if } vlay\_nb = 1 \end{cases} \tag{14.3}$$

where *vlay_nb* is the total number of video layers (base plus enhancement(s)) used by a receiver. The higher this inefficiency ratio, the higher the percentage of time wasted because of the IGMP and multicast routing protocol latencies. Figure 14.6 shows the evolution of this ratio as a function of the VSD and number of video layers produced by the video codec. In our case we are limited to two video layers ($vlay\_nb = 2$) and assume that $igmp\_leave\_lat = 3$ seconds. In this case the *igmp_ineff_ratio* amounts to 10% with $VSD = 60$ seconds, and 5% with $VSD = 120$ seconds. With five video layers, this ratio amounts respectively to 25.0% (prohibitive) and 12.5% respectively.
The IGMP leave latency largely impacts the solution efficiency and using a video segment duration of 60 *seconds is only possible with two video layers* (the common case). Having a higher number of video layers requires to significantly increase the VSD parameter.

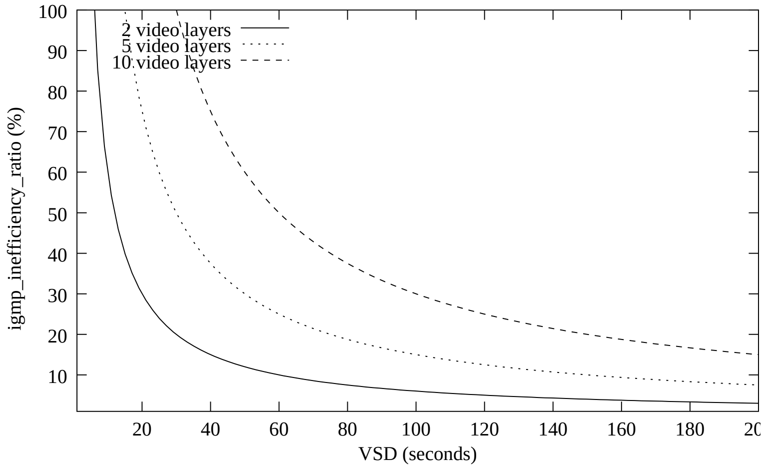


Figure 14.6: Impacts of the IGMP leave latency (igmp_ineff_ratio).

14.3.3 Impacts of the Congestion Control Protocol during Startup Phase

The third aspect to consider is the *congestion control protocol behavior during the startup phase for a given ALC session.* Because of this protocol, the reception rate at a client progressively increases until it reaches a "fair share" (the exact fairness depends on the protocol in use) of the available bandwidth between the source and the client. Depending on the protocol, the time required to reach the steady rate is not so small compared to the VSD parameter and must be considered. We already analyzed and modeled this behavior in chapter 12 and in [74]. This problem affects a client each time he joins a new ALC session, for instance to receive the enhancement video block or when switching to the following video segment. Starting from scratch after joining the following ALC session is the default behavior.

14.3.4 Packet Loss Recovery Capabilities

The VSD parameter has a direct impact on the packet loss recovery capabilities of SVSoA. Losses in the Internet usually occur in bursts, because of router congestion problems or routing instability.

Thanks to ALC's reliability mechanisms (in particular the use of FEC within ALC), these losses can usually be recovered, at least for the base layer which contains the most valuable video information. Intuitively, the longer the video segment duration (VSD), the greater the immunity to losses, and the longer the loss burst that can be recovered. In this section we analyze the SVSoA version 1 robustness assuming that a single burst, of duration *loss_dur*, occurs during a video segment.
The goal of this analysis is to *have an idea on how to initialize the VSD parameter to obtain a certain target robustness*, and what are the other parameters that affect this robustness. The simplification made (single loss burst) does not catch the SVSoA behavior in front of other loss models (e.g. with random isolated losses, or in case of several small loss bursts rather than a single long burst). Yet our scheme also brings some robustness in front of other loss models. The exhaustive analysis of the SVSoA's behavior is left to future works. By default, we only consider the base video layer in this analysis. No guaranty is given for the enhancement layer(s).

Loss recovery capabilities with RLC

This analysis is based on the equations obtained in chapter 12 giving the amount of data transferred after a certain time, when no loss occurs, and taking into account the inefficiency ratio of ALC, *rx_ineff*. In equations 12.1 we can ignore the i factor in front of i^2 (remind that $i * t_0$ is the elapsed time). We can also ignore the non-continuous behavior of the congestion control protocol and use a continuous time instead. We can then write:

$$\frac{Rx(t)}{rx\_ineff} \approx c * t^2$$

where c is a constant. By comparing with equation 12.1:

$$\frac{i^2 b_0 t_0}{3 * rx\_ineff} = c(i * t_0)^2$$

and the value of the constant c is given by:

$$c = \frac{b_0}{3 * t_0 * rx\_ineff} \tag{14.4}$$

Figure 14.7 illustrates the robustness problem when $VSD = 60$ seconds and with a video encoding rate $enc\_rate = 2$ Mbps. Let t_{min} be the time required to receive the amount of video data sent during VSD seconds:

$$\frac{Rx(t_{min})}{rx\_ineff} = VSD * enc\_rate$$

In that case the maximum loss duration is the extra time available at the end of the video segment: $VSD - t_{min}$, and we find (graphically) a value of ≈ 32 seconds.
This is in fact an upper bound and the robustness is largely impacted by the position of the loss burst in the video segment. The maximum recoverable loss period is indeed reduced when the loss period starts in the middle of the video segment, because of the congestion control algorithm which slows down the reception rate after a loss. Depending on the length of the burst, the congestion control algorithm restarts reception at a subscription level j smaller than the subscription level i before the start of the burst: $0 \leq j \leq i - 1$. The worst case where all layers are dropped ($j = 0$), is illustrated in figure 14.8. The maximum recoverable burst length is then only 17 seconds which is now a lower, pessimistic, bound.
Let's now continue with a formal discussion of the problem. Let's t_1 and t_2 be respectively the time before and after the loss burst, of duration t_{loss}. We pessimistically assume that the burst leads the client to drop all layers. We have:

$$VSD = t_1 + t_{loss} + t_2$$

Transmission is successful if:

$$\frac{Rx(t_1) + Rx(t_2)}{rx\_ineff} \geq enc\_rate * VSD$$

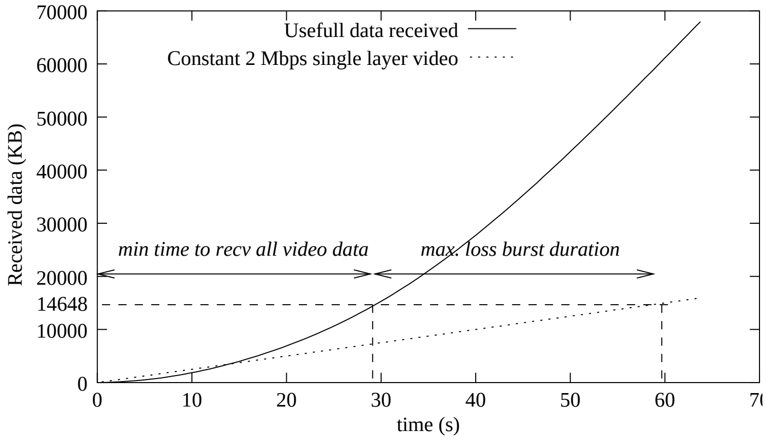


Figure 14.7: Maximum recoverable loss burst length; here the loss burst occurs at the end of the video segment period (60 s).

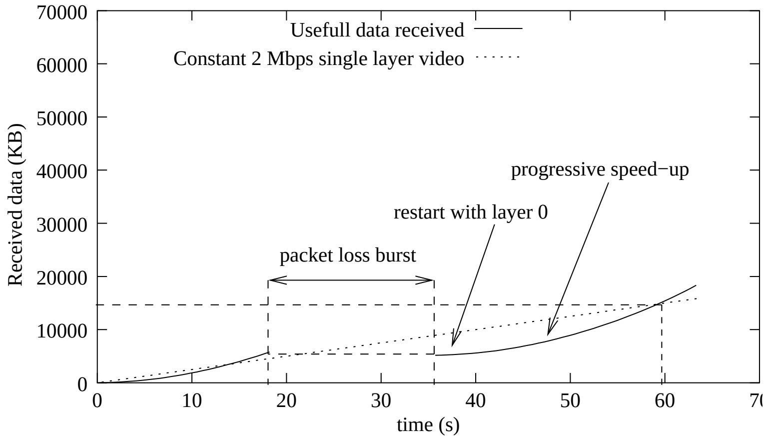


Figure 14.8: Impacts of a loss burst in the middle of the video segment period (60 s).

or:

$$c * t_1^2 + c * t_2^2 \geq enc\_rate * VSD$$

After replacing t_2 and extracting t_{loss}:

$$t_{loss} \leq VSD - t_1 - \sqrt{\frac{enc\_rate * VSD}{c} - t_1^2} = f(VSD, t_1) \tag{14.5}$$

with the following definition interval:
$t_1 \in [0; t_{1\_max} = \sqrt{\frac{enc\_rate*VSD}{c}}]$. The recovery capability is maximum for $t_1 = 0$, then a minimum recovery capability is reached at $t_1 = \sqrt{\frac{enc\_rate*VSD}{2c}}$:

$$t_{loss\_min}(VSD) = VSD - \sqrt{\frac{2 * enc\_rate * VSD}{c}} \tag{14.6}$$

and then the recovery capability increases up to the same maximum obtained for t_{1_max}. Figure 14.9 illustrates the maximum recoverable burst length as a function of VSD and t_1. It uses (eq. 14.4): $c = \frac{160000}{3*0.25*1.66} = 128,000$ bits/sec^2. We find: $t_{loss\_min}(60s) = 16.70$ sec. These curves confirm the high importance of the position of the loss burst in the video segment, and that of the VSD parameter.

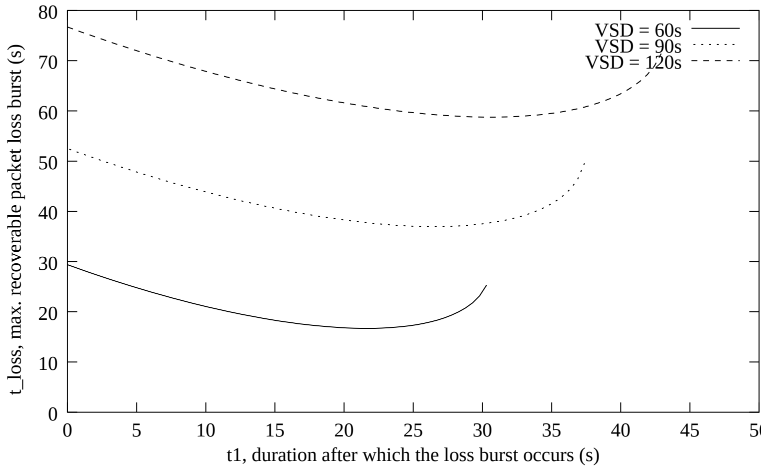


Figure 14.9: Maximum recoverable packet loss burst length when using RLC.

Loss recovery capabilities with WEBRC

With WEBRC, layers are added much faster and SVSoA performs better and tolerates longer loss bursts. Similar to RLC and FLID-SL, We can approximate the behavior of WEBRC (simplify the formula 12.3) by considering the exponential startup behavior of WEBRC and write:

$$\frac{Rx(t)}{rx\_ineff} \approx c * C^t$$

where c is a constant. By comparing with equation 12.3:

$$\frac{C^i * b_0 * epoch}{(C-1) * rx\_ineff} = c * C^i$$

and the value of constant c is given by:

$$c = \frac{epoch * b_0}{(C-1) * rx\_ineff}$$

We now continue our considerations with $C = 4/3$ and $epoch = 0.5$ seconds as proposed in [49]. c is now:

$$c = \frac{3 * b_0}{2 * rx\_ineff} \tag{14.7}$$

Let's now continue with a formal discussion of the problem. Let's t_1 and t_2 be respectively the time before and after the loss burst, of duration t_{loss}. We pessimistically assume that the burst leads the client to drop all layers. We have:

$$VSD = t_1 + t_{loss} + t_2$$

Transmission is successful if:

$$\frac{Rx(t_1) + Rx(t_2)}{rx\_ineff} \geq enc\_rate * VSD$$

or:

$$c * (4/3)^{t_1} + c * (4/3)^{t_2} \geq enc\_rate * VSD$$

After replacing t_2 and extracting t_{loss}:

$$t_{loss} \leq \frac{ln(\frac{enc\_rate * 3^{t_1} VSD - c * 2^{2t_1}}{c}) - 2 * t_1 * ln(3/2) + VSD * ln(3/4)}{ln(3/4)} \tag{14.8}$$

with the following definition interval:
$t_1 \in [0; t_{1\_max} = \frac{ln(\frac{c}{enc\_rate*VSD})}{ln(3/4)}]$. The recovery capability is maximum for $t_1 = 0$, then a minimum recovery capability is reached at $t_1 = \frac{ln(\frac{2*c}{enc\_rate*VSD})}{ln(3/4)}$:

$$t_{loss\_min}(VSD) = \frac{ln((\frac{c}{enc\_rate*VSD})^{\frac{2*ln(2)}{ln(3/4)}})}{ln(3/4)} - \frac{2*ln(\frac{c}{enc\_rate*VSD})*ln(3/2)-(VSD*(ln(3/4))-2*ln(2))*(ln(3/4))}{ln(3/4)^2} \quad (14.9)$$

and then the recovery capability increases up to the same maximum obtained for t_{1_max}. Figure 14.10 illustrates the maximum recoverable burst length as a function of VSD and t_1. It uses (eq. 14.7): $c = \frac{3*160000}{2*1.66} = 144578$ bits/sec. With the same parameters as with RLC, we find: $t_{loss\_min}(60s) = 18.09$ sec which can be compared with the 16.70 sec of RLC. The benefits of WEBRC compared to RLC increase when VSD increase.

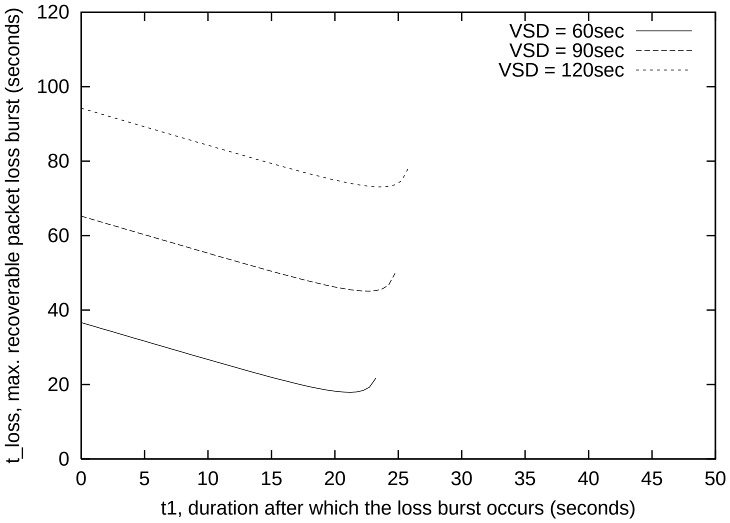


Figure 14.10: Maximum recoverable packet loss burst length when using WEBRC.

14.3.5 Transmission Rate of the ALC Session

Another important parameter is b_0, the transmission rate of the base ALC layer. A higher b_0 leads to a faster reception in the startup phase, linearly with RLC, in b_0^2 with FLID-SL. But a high b_0 also limits the possibilities to serve low end receivers. Therefore the following aspects must be considered:

- Required reception time in front of the video encoding rate: The higher the video encoding rate, the larger b_0 should be.
- The target environment (closed network, Internet,...):
  This target environment defines the possible heterogeneity of the users and their access networks. When serving Internet clients, b_0 should be compliant with the slowest possible client.

14.3.6 Initializing SVSoA version 1 in Practice: a Summary

In practice SVSoA can be correctly initialized according to the following steps:

step–1: Retrieve the average encoding rate enc_rate and number of layers of the video, $vlay\_nb$ (usually 2).

step–2: if $vlay\_nb \geq 2$, then use $VSD = 30 * vlay\_nb$ to keep the *igmp_ineff_ratio* constant and equal to 10%, assuming an IGMP latency of 3 seconds (equation 14.3). if $vlay\_nb = 1$, then IGMP has no effect, and using $VSD = 60$ seconds for instance is appropriate.

step–3: With RLC, retrieve the t_0 parameter, and with WEBRC the *epoch* parameter. Estimate the reception inefficiency of ALC and its FEC code (e.g. we use 1.66).

step–4: Define the transmission rate on the base ALC layer, b_0. This choice depends on the target environment since this is the minimum reception rate.

step–5: Calculate the minimum loss burst immunity for the base video layer. With RLC use equation 14.6:
$t_{loss\_min} = VSD - \sqrt{\frac{6 * enc\_rate * VSD * t_0 * rx\_ineff}{b_0}}$. With WEBRC, use equation 14.9. If this value is judged too low, it means that the video bit-rate is too high compared to the transmission rate. So increase the b_0 value and reiterate at step 4. Alternatively we can set a higher VSD and reiterate step 5.

Many parameters are only approximations. For instance the video encoding rate is only an average (see figure 14.11), and the *rx_ineff* is not known in advance. This is not a major issue yet since several equations of previous sections (e.g. $t_{loss\_min}(VSD)$) are for the worst case and rely on pessimistic assumptions (e.g. the client drops all the layers after the loss burst).

14.4 Experimental Results

14.4.1 Experimental Conditions

The Streaming Approaches Compared

We implemented our proposal using our MCLv3 library that implements the ALC protocol [93], and the MPEG4IP [3] MPEG-4/RTP application.
We compare the SVSoA performances with that of a "classical" FEC based streaming scheme inspired from [99]. It uses a *single* ALC session, each video layer is mapped onto a distinct ALC layer, and it uses RLC for congestion control, like SVSoA. The sender sends each frame sequentially. Reed-Solomon FEC protection is used for the base video layer, using the same parameters as in [99] (30 FEC packets are added to each block of 225 source packets). The enhancement video layer is sent without FEC protection, since it has less importance. Adding proactive FEC undoubtedly consumes additional bandwidth, but it enables a fair comparison with SVSoA which heavily relies on FEC. Interested readers can refer to [70] for additional comparisons with a classical streaming scheme which does not include any FEC protection. Results are globally similar to that obtained with the "classical" FEC based scheme.

Video Encoding and SVSoA Parameters

Our tests use a 120 second video, encoded with a constant 25 fps frame-rate, and using a spatial scalability encoding which provides two video layers, a base layer and a single enhancement layer. The bit-rate of both layers is on *average* 667 kbps respectively, but the instantaneous bit-rate fluctuates around this average, as shown in figure 14.11 (this figure represents the sum of the two video layers bit-rate).
In practice a dedicated MPEG-4 decoding hardware would be used by clients for real-time decoding. Since this facility is not available in our testbed, the video is encoded off-line, and the decoding/playing features at a client are turned off. We thus focus on transmission aspects and avoid any interference with CPU intensive tasks.
SVSoA is initialized with $b_0 = 131$ kbps and $VSD = 60$ seconds. The theoretical minimum loss recovery capability is $t_{loss\_min} = 21.0$ seconds.

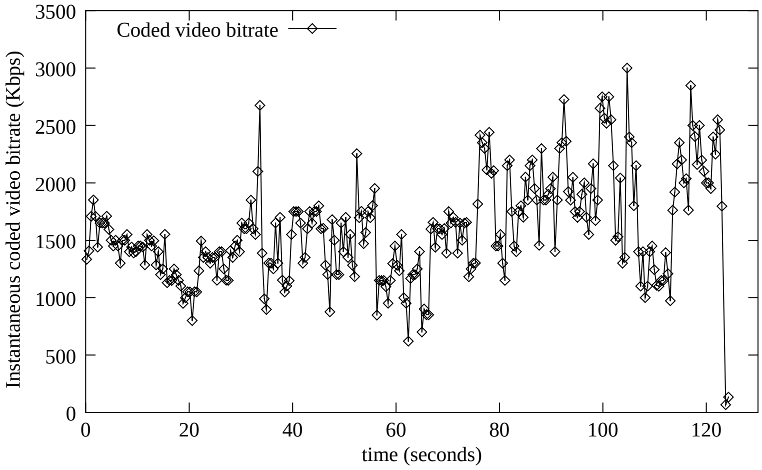


Figure 14.11: Instantaneous video bitrate of the two layers.

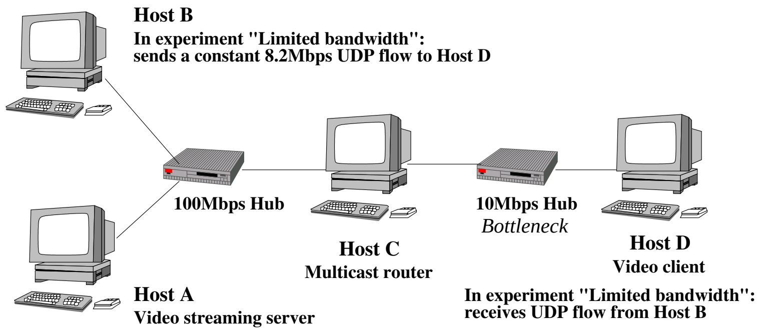


Figure 14.12: Our testbed.

Testbed and Scenarios

Our testbed (figure 14.12) is composed of four PC P-III/Linux attached to two different Ethernet subnets. The right subnet has a limited bandwidth of 10 Mbps. Host C in the middle is the multicast router and uses the `mrouted` multicast routing daemon. Host A is the streaming server and host D the client. Two scenarios are considered:

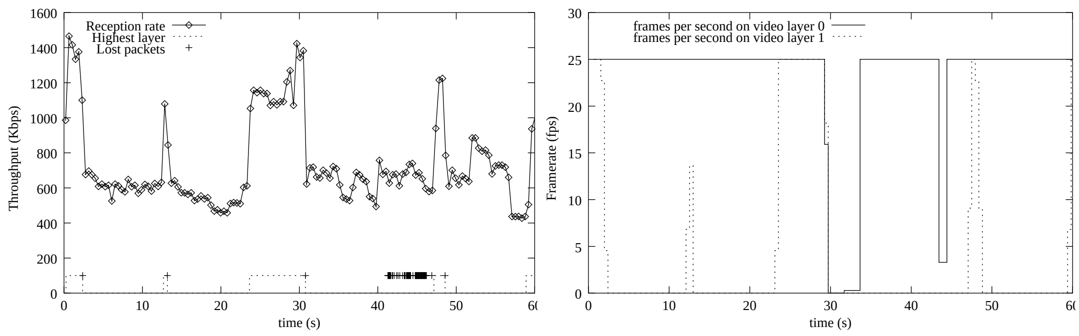
1. *Limited bandwidth* to evaluate both streaming approaches in presence of a concurrent network flow, in a limited bandwidth environment. During this test, host B sends a constant bit-rate 8.2 Mbps UDP flow to host D. Host D receives both the UDP flow from B and the video from A.

2. *Long burst of packet losses* to evaluate the error recovery capabilities of both approaches. 10 seconds after the start of the video transmission, we unplugged the network cable during 15 seconds, and reconnect it up to the end of the test. No background traffic is used in this test.

14.4.2 Experimental Results

Behavior in a Limited Bandwidth Environment

With the classical approach, the transmission rate for each video layer is highly dependent on the instantaneous video bit-rate. A 255/225 Reed-Solomon FEC encoding operates on 1-2 seconds chunks of frames within the video. Therefore the transmission rate is constant for these short periods. When the cumulative rate on both layers exceeds the available bandwidth (e.g. after 3 seconds in figure 14.13 (a)), packets get lost. According to RLC, the client then drops a layer, which reduces the video quality, in order to adapt to the available network bandwidth. Some time after 12 seconds, the



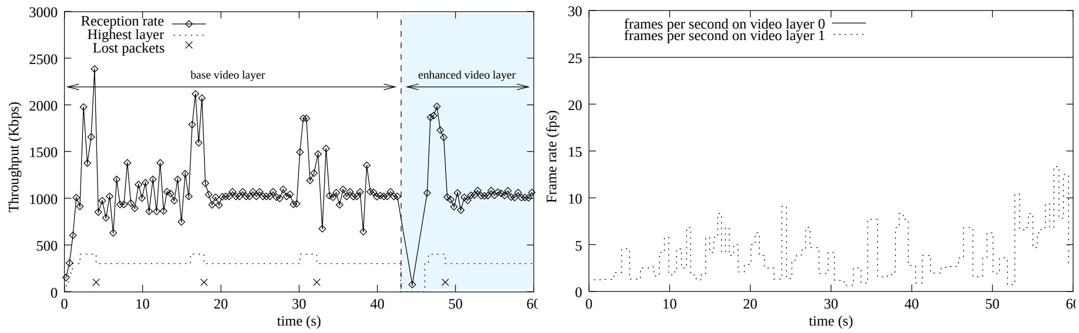
(a) Reception rate

(b) Frame rate

Figure 14.13: Tests with limited bandwidth and the classical FEC based approach.

enhancement layer is once again added, which triggers losses, and the client drops the higher layer one more time. We see that the rough granularity provided by the spatial scalability encoding does not enable an efficient behavior. We also see that losses may affect both video layers (e.g after 30 seconds or after 43 seconds (figure 14.13 (b))). FEC protection on the base video is obviously not enough at these moments. Each loss on the base layer prevents the client to decode the associated frame on the higher layer if received which amplifies the phenomenon.

This is not the case with SVSoA where the reception rate is only determined by the congestion control protocol, independently of the video bit-rate (compare figures 14.13 (a) and 14.14 (a)). Transmissions are smoothed over the time which is an advantage from a networking point of view. Figure 14.14 (b) shows that the frame rate of the base layer is constant during the whole test, and a minimum video quality is provided to the client. However the frame rate of the enhancement layer is relatively low since the spare bandwidth is not sufficient to receive more frames.



(a) Reception rate

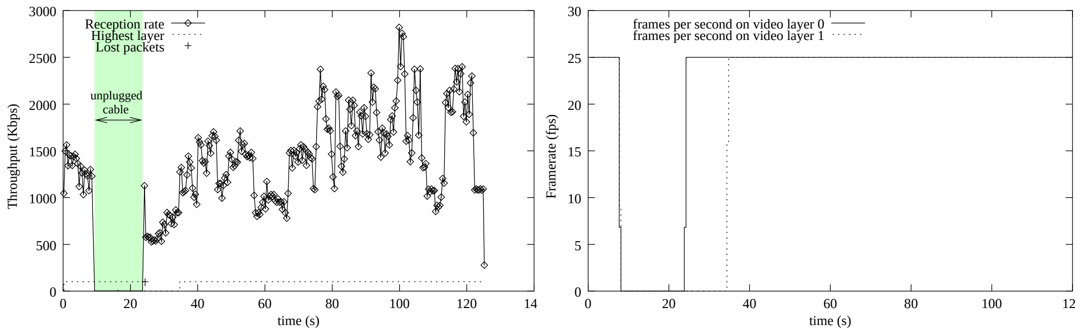
(b) Frame rate

Figure 14.14: Tests with limited bandwidth and SVSoA.

Behavior With a Long Burst of Packet Losses

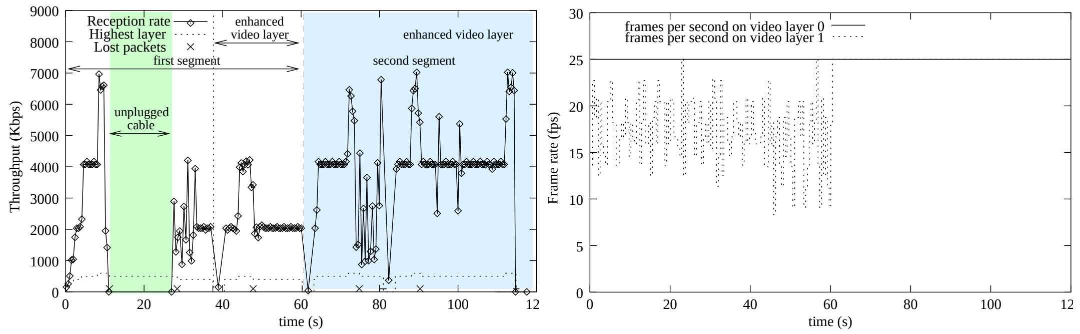
The reception for both approaches is totally interrupted while the cable is unplugged (between $t \approx 10s$ and $t \approx 25s$ in figures 14.15 (a) and 14.16 (a)). With the classical approach, the reception restarts with the base video layer only, the enhancement layer being only added after some time. With SVSoA the reception of the base layer continues after the burst, until completely received.

The frame rate of the resulting video at the client shows a big hole on both video layers with the classical approach (figure 14.15 (b)). No frame can be displayed while the cable stays unplugged. For



(a) Reception rate (b) Frame rate

Figure 14.15: Tests with a 15 second loss burst and the classical FEC based approach.



(a) Reception rate (b) Frame rate

Figure 14.16: Tests with a 15 second loss burst and SVSoA.

the enhancement layer, this time is longer since the ALC layer is only added after an additional delay (RLC behavior). FEC protection cannot do much in front of such a long burst of losses.
With SVSoA the client receives the whole base video layer, thereby assuring a minimum video quality. The frame rate of the enhancement video layer is not optimal though because the client needs more time to reconstruct the base layer, which leaves less time for the enhancement layer. But the lost frames of the enhancement layer are spread equally over the whole segment, which provides a *globally constant video quality*.

14.4.3 Discussions

In the two selected scenarios, SVSoA largely outperforms the classical FEC based streaming approach. This classical approach, which serves as a reference, is obviously not optimal, and could benefit from several improvements, like: (1) the use of a QoS service to protect the base layer, (2) the use of a finer video scalability granularity, and (3) a really constant bit-rate video encoding. Yet SVSoA assumes none of these features, which are three major, very restrictive assumptions.
The proactive FEC protection used is not sufficient to reconstruct all losses of the base video layer. Adding more FEC is not the solution since it increases the consumed bandwidth, leading to more losses. All solutions requiring some feedback information from the receivers to adjust the amount of proactive FEC create scalability problems. Besides the ideal amount is not the same for all clients, all the time, and defining several parallel streams is then unavoidable. This is why we chose to use the FEC scheme proposed by [99].
The experiments show that SVSoA behaves well in a congested environment, no matter how many

layers the video codec produces. Congestion control enables to adapt to the available bandwidth. Because of the natural buffering capability of SVSoA, packet losses, even in case of long bursts, do not automatically lead to frame losses and sudden video quality changes. They only increase the time spent receiving a given layer, and thus reduces the probability of receiving the enhancement information entirely. And because of the high randomness of transmissions introduced in an ALC session, the video quality is almost constant during each VSD period.
Even if the experiments carried out involve a single client, the SVSoA scalability is guaranteed by the total absence of feedback packets. Scalability is only limited by multicast routing protocols.
The SVSoA performances experienced are limited by some sub-optimal solutions in the ALC implementation used [70]. More recent congestion control protocols (WEBRC), more efficient large block FEC codes, and the meta-object FEC encoding we introduced in Chapter 11 would largely improve the performances. The idea of meta-object encoding lead us to design SVSoA version 2.

14.5 SVSoA version 2

SVSoA version 1 introduces a large playout latency, around 60 seconds long (see [70, 73] for a detailed justification). Therefore, if it is well suited to the large scale distribution of videos or TV programs over the Internet, it cannot be used for interactive applications like video-conferencing, because of its significant playing delay. Besides SVSoA version 1 relies on multicast routing to make sure clients receive the block associated to the base video layer first before trying to get the enhancement layer. This is meaningless in a wireless network where all packets will be anyway broadcast to all clients, no matter whether they belong to the base or enhanced video layer.
In this section we introduce version 2 of SVSoA which solves these two limitations, making it a robust streaming solution, in particular, and we focus on this use case, in wireless environments.
SVSoA version 2 is therefore well suited to wireless video broadcast, but it also performs well over traditional wired networks and in particular the Internet.

14.5.1 Scenarios

As SVSoA version 1, SVSoA version 2 is suited to traditional routed environments. Indeed, in a routed wired network like the Internet, congestion control is of utmost importance to avoid congestion within the network and to support client heterogeneity, and the congestion control building block of ALC perfectly addresses these issues.
However, in a wireless environment congestion control is not an issue since (1) the maximum capacity of the channel is known (even if the actual capacity largely fluctuates) and (2) no routing is performed. Losses on a wireless channel are due to interferences and not to router congestion. The ALC congestion control building block should therefore be switched off in that case, and constant rate transmissions, using a single layer (i.e. multicast group) be used instead (this is the default working mode of ALC in 3GPP and DVB-H environments). Since SVSoA version 1 largely relies on the congestion control building block of ALC and since it uses several ALC sessions to broadcast the content, it is less adapted to the wireless case.
SVSoA version 2 is also adapted to the case where the streaming server streams the video through the Internet to both wired clients and to wireless clients (figure 14.17). To that purpose, a dedicated SVSoA relay has to be placed at the border of the wireless network. The SVSoA relay forwards the streamed video from the wired network to the wireless network, by sending the content without any congestion control. The SVSoA relay knows the bandwidth supported by the wireless channel and will send the video at a compatible bitrate. On the wired network side the SVSoA relay receives the content using the congestion control protocol. It may receive on a higher rate than the one supported by the wireless network in order to be immune to packet losses on the wired network side.

14.5.2 Principles

SVSoA version 2 differs from version 1 mainly by the fact that only one ALC session is used to transmit the video content. SVSoA version 2 does not exploit the possible hierarchical video encoding, but it can however offer an adapted video quality to each receiver. This is a major improvement since it

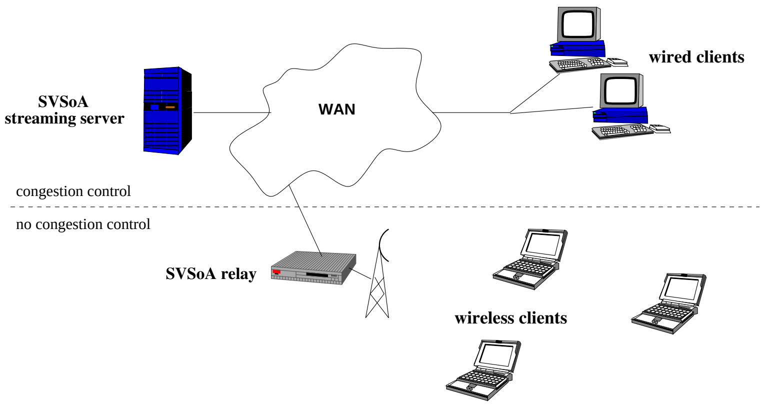


Figure 14.17: SVSoA version 2 scenario.

removes constraints (hierarchically encoded videos) and many transmission inefficiencies specific to version 1 (e.g. the switching time between the two ALC sessions at a receiver is largely responsible of the high playing delay). The playing latency of the version 2 approach is now significantly reduced (typically divided by three), and is now essentially related to the desired robustness. At the same time, SVSoA version 2 keeps the main assets of version 1, namely: (1) massive scalability, (2) TCP friendliness over routed networks, (3) client heterogeneity support, (4) and immunity to long bursts of packet losses.

Sender and receiver behavior

As in SVSoA version 1 the sender partitions the video into segments of equal length. The video segment period is a parameter that has to be adapted to the video features, the given environment and the robustness to losses that is required. The video segment duration is significantly smaller in SVSoA version 2 than in version 1. We were able to use values of about 20 seconds with very satisfying results (see section 14.5.4), compared to 60 seconds with version 1.
Each video segment is then FEC encoded and transmitted in a carousel transmission mode over the entire video segment period (figure 14.18).

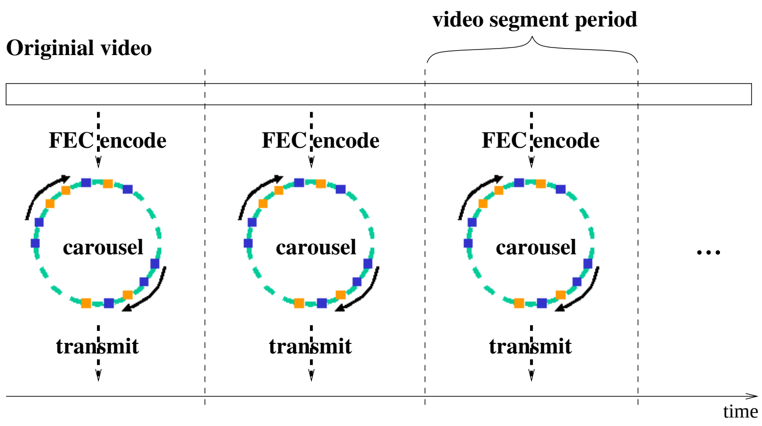


Figure 14.18: SVSoA version 2 sender behavior.

When a new receiver joins the SVSoA session, he joins the associated ALC session where the current segment, n, is sent. He cannot display anything at this moment. When the transmission of segment $n+1$ starts[1], the receiver begins to play the video of segment n. Therefore a receiver always plays the

[1]Receivers must be synchronized on segment boundaries. This is made possible by dedicated control packets sent regularly by the source as in SVSoA version 1.

previous video segment while receiving the current one, which of course introduces a playing latency of one video segment period (figure 14.19).

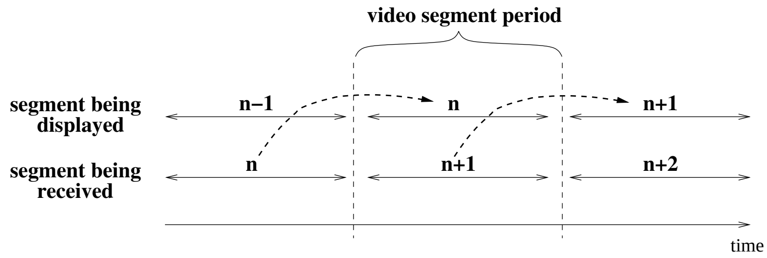


Figure 14.19: SVSoA version 2 receiver behavior.

Using a FEC partial reliability service

The FEC decoding process of one segment may not end successfully if not enough packets have been received by the receiver (i.e. less than k or a little bit more). This is the case when reception is interrupted, like at the end of each video segment period, or if the reception is affected by too many losses (e.g. if a vehicle enters an environment with many obstacles that only enables an erratic connectivity). A partial reliability service should maximize the amount of original content reconstructed at a receiver, even if FEC decoding did not finish. In chapter 11 we analyzed the inherent capabilities of each FEC code to maximize this partial decoding, and identified how to efficiently use the FEC codes in this context. These results are directly used in SVSoA version 2, by using a partial reliability service for each FEC encoded video segment.

This service allows, *even if the current segment has not been entirely received*, to: (1) Stop FEC decoding at the end of a segment, (2) benefit from already decoded or received packets of the segment, and therefore (3) display the segment in an acceptable and stable (since losses are spread over the whole period) video quality.

As shown in chapter 11, the best results are obtained using a Reed-Solomon FEC code. The structure of the video segments (i.e. the position and length of each video frame within the video segment) is transmitted with an in-band signaling mechanism in dedicated signaling objects. The detailed description of the signaling mechanism is out of scope of this work.

14.5.3 Robustness Evaluation

In this section we analyze the robustness that SVSoA version 2 provides, i.e. the amount and the type (i.e. in burst or single losses) of supported losses.

Impacts of the transmission scheduling

Why use a randomized carousel transmission? SVSoA version 2 is aimed to be adapted to very heterogeneous environments as we have seen in section 14.5.1. The loss patterns vary strongly from one environment to another (e.g. wireless channel compared to the Internet). Even if we only consider the wireless channel the loss behavior may be quite different from one receiver to another (e.g. a car that regularly crosses tunnels compared to a person that stays at home). In order to achieve good performances with all these loss patterns we chose to use a randomized carousel transmission, i.e. a carousel transmission where the packets of one segment are sent in a completely random order rather than sequentially (e.g. all source packets first and then all parity packets) for instance. For some specific loss pattern another very specific packet scheduling may perform much better, but this same scheduling will have catastrophic performances when another loss pattern occurs. With a totally random transmission scheme we provide a service that achieves always the same performance, independently of the loss model, as we have shown in chapter 9

Why use FEC? The advantage of using FEC in that context is not that obvious. Rather than using FEC we may simply retransmit the content twice (or more), and a receiver could then receive missing packets before the end of one segment period. Does FEC really increase performances compared to several retransmissions? To answer this question we made a simple simulation: Given a content of size $k = 6000$ we first transmit the content twice, all $2*k = 12000$ packets being sent completely randomly, and look at which moment (i.e. the number of received packets) the content has been entirely received. We compare this with an approach using FEC: the content is encoded using Reed-Solomon and FEC expansion ration of 2.0, and all packets (source and parity) are sent completely randomly. Results are shown in figure 14.20. We see that with RSE the number of packets needed to entirely reconstruct the content is much smaller than when using no FEC at all. With no FEC the receiver nearly needs to wait until the end of transmission to entirely get the content. Losses have not been taken into consideration in this simulation. However since packets have a higher correction power when FEC is used than without FEC, it is clear that results would have been similar with losses. Therefore this simple simulation shows us that FEC clearly should be used in SVSoA version 2.

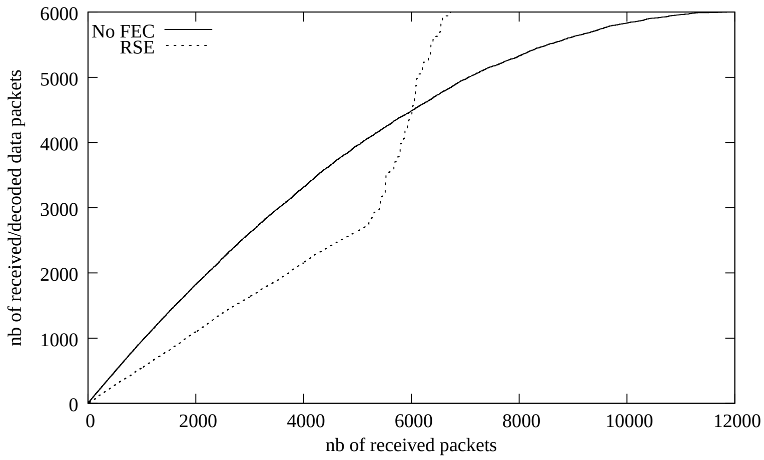


Figure 14.20: Reed Solomon versus no FEC using a randomized transmission.

Amount of tolerated losses

We now discuss the robustness to transmission errors of our approach, i.e. the amount of tolerated packet losses depending on some parameters. We limit our evaluation to the case where we broadcast without any congestion control (i.e. the wireless case). We can completely abstract from the actual loss pattern (in burst or single packet losses) since transmission are made in a completely random manner as explained before.

Lets assume that we want to stream a video that has been encoded with a bitrate of $video\_bitrate$ kbps. Depending on the maximum supported bandwidth $max\_bandwidth$ of the wireless channel, we can produce parity packets using a FEC ratio fec_ratio, with $max\_bandwidth \geq fec\_ratio * video\_bitrate$. Here we only consider the case where the transmission rate of the ALC session is

$$tx\_rate = fec\_ratio * video\_bitrate$$

, i.e. that during one segment duration VSD all source and parity packets are only send once (and all packets are sent randomly during the whole segment duration).

The usage of FEC introduces an inefficiency ratio $inef\_ratio$ that needs to be evaluated experimentally, as we have done it in chapter 10. The receiver needs to receive at least $video\_bitrate * VSD * inef\_ratio$ kbit per segment in order to display the video with full quality for that segment. That means that the amount of supported losses is:

$$\begin{aligned} & supported\_losses \\ = \; & tx\_rate * VSD - video\_bitrate * VSD * inef\_ratio \\ = \; & video\_bitrate * VSD * (fec\_ratio - inef\_ratio). \end{aligned}$$

The supported loss rate for one segment is then:

$$\begin{aligned} supported\_loss\_rate &= \frac{supported\_losses}{tx\_rate * VSD} \\ &= 1 - \frac{inef\_ratio}{fec\_ratio}. \end{aligned}$$

Lets have an example: Suppose that we want to transmit a video over a 802.11 wireless channel. The video is encoded at $video\_bitrate = 1.2$ Mbps using a FEC expansion ratio of $fec\_ratio = 2.0$ (therefore $tx\_rate = 2.4$ Mbps). This leads to $inef\_ratio \approx 1.12$ (evaluated experimentally in chapter 10). The supported loss ratio is then $supported\_loss\_rate = 1 - \frac{1.12}{2.0} = 44\%$. Having a segment duration of $VSD = 20$ seconds, this means that we support $supported\_losses = 8.8$ seconds of losses (either as burst or as single packet losses).

Impacts of Partial Reliability

The usage of a partial reliability service allows to support even longer periods of packet losses. All losses above the threshold $supported\_loss\_rate$ presented before will now affect the video quality.
We evaluated the period of losses that affects video quality in a simple simulation. Lets take the above example again: Assume that the video has a frame rate of 30 frames per second, that means that each segment has 600 video frames. With a packet size of 512 bytes there are 12000 packets to transmit (since $1.2\ Mbps * 20\ sec/512\ bytes = 12000$).
Figure 14.21 shows that video quality adaptation is made between $\approx$ 5100 received packets (this corresponds to $\approx$ 8.5 seconds of reception time) and $\approx$ 6700 received packets (this corresponds to $\approx$ 11.2 seconds reception time).
That means that additionally to the $supported\_losses = 8.8$ seconds of supported losses we have a period of 3.7 seconds where the video quality is adapted. This period would not exist if no partial reliability service had been used.
Lets assume that we have a loss rate of 50%. In this case we can display 274 frames out of 600 for that segment. Supporting such a high loss rate may be very interesting especially in the wireless case.

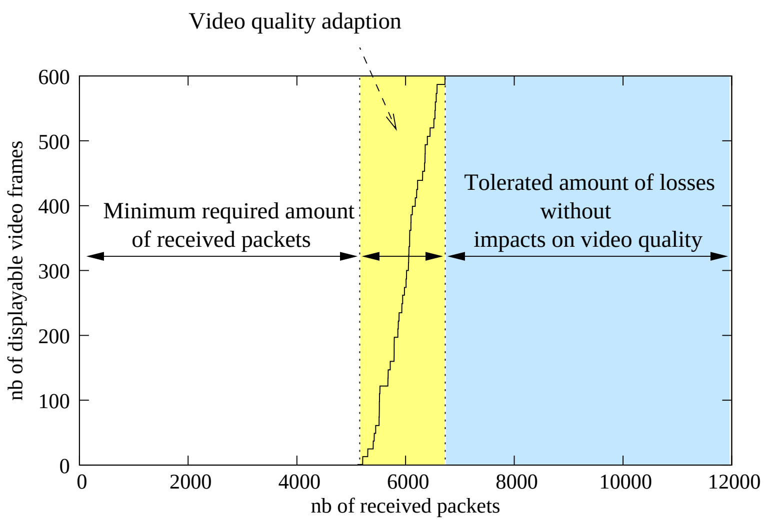


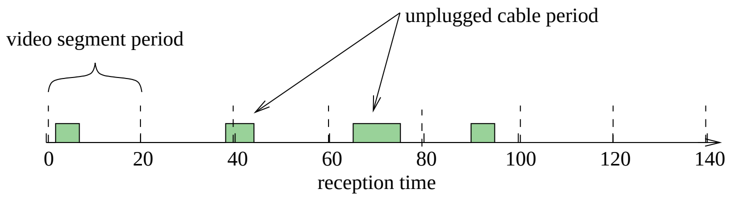
Figure 14.21: Displayable video frames.

14.5.4 Experimental results

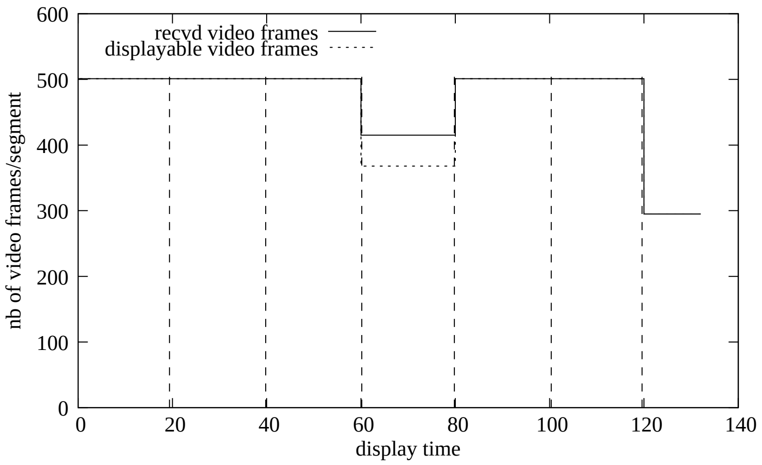
We implemented SVSoA version 2 using the open-source implementation MCLv3 [93] of ALC. We tested the case of intermittent connectivity and therefore totally interrupted the reception by unplugging the network cable over several seconds. We sent a single layered variable bit rate (aver. bit rate is 654 kbps) video, with a frame rate of 25 fps, using SVSoA version 2 without any congestion control over a Local Area Network. We used a FEC ratio of 3.0. The video lasts $\approx$ 132 seconds and the segment period duration VSD is 20 seconds. That means that 500 frames are transmitted within each segment (except for the last one with only 295 frames). To simulate intermittent connectivity we unplugged the network cable several times for different durations as follows (and as shown in figure 14.22(a)):

1. Loss period 1: At $t = 2$ seconds we unplugged the cable for 5 seconds.
2. Loss period 2: At $t = 38$ seconds we unplugged the cable for 6 seconds.
3. Loss period 3: At $t = 65$ seconds we unplugged the cable for 10 seconds.
4. Loss period 4: At $t = 90$ seconds we unplugged the cable for 5 seconds.

The results, i.e. the number of received and displayed frames, are shown in figure 14.22(b).



(a) Simulated loss bursts.



(b) Received and displayed video frames.

Figure 14.22: Experimental setup and results.

We see that unplugging the cable for 5 or 6 seconds (loss periods 1, 2 and 4) had no impact on video quality, since the number of displayed and received frames is 500 in the corresponding segments, i.e. all frames have been received. Also loss bursts that straddle two video segments (loss period 2) is handled correctly.

In segment starting at $t = 60$ seconds the number of received frames is less (415) than the maximum. The loss burst (loss period 3) was too long, and the receiver was not able to recover the content of that segment entirely. However thanks to partial reliability most of the frames of that segment have been reconstructed. Because of frame inter-dependence not all of the 415 received frames can be displayed: only 368 frames out of them can be displayed properly. Error concealment techniques are not considered here, i.e. a frame that cannot be displayed because of inter-frame dependence is considered as completely lost. When playing the video we notice that some frames are missing in the corresponding segment, but overall quality is very satisfying.

With the same video SVSoA version 1 there is no video quality adaptation (since the video is not hierarchically encoded). Since no congestion control is required, and since there is only one base video layer SVSoA version 1 uses the same parameter as version 2 in this context. The only difference between version 1 and 2 in this case is the usage of partial reliability. Therefore the segment starting at $t = 60$ seconds would have been entirely lost. The video would have stopped playing for 20 seconds. More classical approaches like in [4] or the one used in our experiments in section 14.4 would also had problems scoping with the "short" (5 or 6 seconds) loss bursts, since the buffering time is much shorter and more important the FEC capabilities are less important. No video at all would have been displayed during these periods as we have already shown in section 14.4.

14.6 Conclusions

This chapter introduces a novel multicast streaming solution, SVSoA, for hierarchically encoded videos. Our solution is in fact mid-way between reliable multicast file transfer and streaming. It directly benefits from the ALC reliable multicast protocol assets in terms of unlimited scalability, congestion control (TCP-friendly behavior), and error recovery (packet bursts are easily recovered). Thanks to ALC, our approach addresses the client heterogeneity issue and lets each of them experience the best possible display made possible by their access network. Our approach limits video quality instability by smoothing the effects of packet losses over periods of 20 seconds to one minute. Finally, the solution is immediately deployable since it does not assume the presence of any specific service (like QoS support) within the network.

We explain how to initialize the various parameters of SVSoA version 1, what are the associated trade-offs, and provides answers to several issues like the IGMP leave latency or the impacts of the slow startup phase of the congestion control protocol. We implemented our approach and carried out a set of experiments on a local testbed. Results shows that SVSoA version 1 behaves appropriately, in line with the theory.

Finally, this work introduces Scalable Video Streaming over ALC (SVSoA) version 2 for the video broadcast over wired or wireless channels. As SVSoA version 1 it benefits from all the ALC reliable multicast protocols assets in terms of unlimited scalability, congestion control, error recovery etc. It is no longer restricted to hierarchical encoded videos and can offer an adapted video quality to each receiver thanks to an FEC partially reliability service.

Chapter 15

Conclusions

This thesis deals with multicast content delivery protocols and applications. It focuses on the FLUTE application and the underlying ALC transport protocol because of their increasing importance, especially in the mobile world, after their adoption in the 3GPP MBMS and DVB-H IP DataCasting (IPDC) standards.
New generations of mobile devices will undoubtedly dominate the future Internet, bringing millions of new customers, and above all bringing broadcast and multicast technologies to the cellulars. These technologies have therefore the opportunity to be widely deployed, not only on dedicated cellular networks, but also more generally in the Internet, where inter-domain multicast routing is far behind expectations. Reliable multicast content delivery protocols will therefore quite likely gain a lot of importance.
Another opportunity for reliable multicast is the broad availability of high bandwidth Internet access technologies (e.g. ADSL/ADSL2) offered to the public. Download of very popular contents on these networks is done mainly using peer-to-peer solutions today, but these solutions are clearly suboptimal for many use cases, and reliable multicast solutions can, once the multicast routing structure has been deployed, favorably replace peer-to-peer techniques (at least for some specific use cases).

15.1 Contributions

Reliable multicast relies on several building blocks such as congestion control, Forward Error Correction and security, and can be adapted to each use case (e.g. cellular network or high bandwidth access network). Of particular interest in this thesis is the FEC building block, which is a key component for any scalable content delivery solution. The work can be divided into two parts:

1. - Reliable multicast building blocks:

In particular I looked at:

- *Forward Error Correction Building Block:* I especially examined the broad class of "Low Density Parity Check" (LDPC) large block FEC codes, and came to the conclusion that these codes have a very high potential. Therefore all our work turns around this class of codes.

  I especially looked at its two derivates LDGM-Staircase and LDGM-Triangle, the latter had been designed during this thesis. I benchmarked these codes in detail with respect to their inefficiency ratio (i.e. error correcting capacities), memory requirements and decoding/encoding speeds, and compared them to the RSE small block code.

  The implementation of the LDPC codes is also one of my main contributions, and I largely optimized it during this thesis. I found that LDPC codes and their implementation, have very promising performance especially when used with large files.

The FEC building block has to be adapted to the losses it counters, and I experimentally evaluated how to best schedule packets, in a reliable file delivery application. Based on these results I established recommendations on how to best use the FEC building block with respect to the encountered losses.

Finally, I analyzed the codes with respect to their ability to offer a partial reliability service, i.e. their ability to decode parts of the content even if the whole process cannot finish because too many packets are missing. I found quite counter-intuitive results: in spite of the iterative decoding algorithm the class of LDPC codes is less adapted for a partial reliability service, and a small block code as RSE should be used instead.

- *Congestion Control Building Block:* I looked at the startup behavior of the three congestion control protocols RLC, FLID-SL and WEBRC. I show that the startup phase of these protocols significantly impacts the performance of a file download application. I mathematically modeled the behaviors which allows to predict the required download time.

2. - Reliable multicast applications:

I contributed to:

- *FLUTE implementation and extensions:* Based on our previous results I proposed a file aggregation mechanism for FLUTE at the IETF.

  Its purpose is (1) to provide a signaling mechanism that logically groups a set of interdependent objects and (2) to improve transmission performance, by FEC encoding the aggregated object rather than individual ones. This is efficient because LDPC codes are better adapted to large objects than RSE small block codes, and it helps eliminating the "coupon collector problem". Moreover this scheme uses the assets of the partial reliability service that I presented before.

  I implemented the FLUTE file delivery application/protocol in our MCLv3 library, which also includes the ALC protocol. The implementation is quite powerful and allows fine control of FLUTE (implemented as a general purpose library), ALC and the associated FEC codes. The library has become quite popular and commercial products use it.

- *Video Streaming:* I also looked at large scale video delivery: Designing a video streaming solution for an environment having no or a constrained back channel, due for instance to scalability requirements, leads to many challenges. I introduce the SVSoA version 1 and version 2 approaches, based on ALC and using a streaming technique close to the broadcast of files as used in FLUTE. The first one requires hierarchically encoded videos, while the latter is more adapted to a wireless broadcast use case and does not need hierarchical encoded videos anymore. These approaches benefits from all the ALC reliable multicast protocols assets in terms of unlimited scalability, congestion control and error recovery.

15.2 Future work

The issues I explored in this thesis can be further enhanced and extended. Among these enhancements I envision:

15.2.1 Application Level FEC Schemes

Application level FEC will become more and more important, and its use won't be restricted to the reliable multicast use case described in this thesis. To cite only a few examples that strongly rely on FEC: peer-to-peer approaches (e.g. [39, 21]), network storage with approaches like OceanStore [90] and the quite new area of network.

Further Comparision Studies

To achieve the best results in these new fields of applications, it is very important to have a very good knowledge of all the existing application level FEC codes and their respective performances. Therefore the comparision study of the large block LDPC codes carried out in this thesis should be extended, by benchmarking the actual performances of rateless codes such as Raptor and Online-codes. Other codes, that have not been not mentioned in this work, like algebraic-geometric codes [12] and SPC product codes [36] should also be included in this extended comparision study.

FEC for Embedded Systems

Moreover, FEC codes will increasingly be used by constrained devices like cell phones. There is therefore a strong incentive to adapt the existing codes and codecs to the specificities of embedded systems, i.e. small working memory and small CPU consumption. New codes, well suited to these environments, may arise, as well as new techniques to design embedded FEC codecs.

New Rateless Codes

An interesting challenge is to find new alternative rateless/expandable codes. These codes have very interesting properties, and the fact that the code can create an infinity of parity symbols is an important asset in many use cases. Having an IPR free rateless code would be of great help to the community. Because of the existence of many IPRs in this area it is quite difficult to design new expandable codes that are not covered by these patents. I however still believe that such a scheme exists.

Other Improvements

More generally in the area of FEC, even if a lot of work has already been carried out, there remain many things to explore. I envision the following directions:

1. addressing the performance issue with large block FEC codes when used with small files. Using smaller symbols as it is often proposed, is one possibility but not the only one. Plank et al. [89] did first evaluations and observations in this domain by experimentally exploring a large set of LDPC codes, that have been applied to small blocks. Further studies are needed to find and understand optimal codes for small blocks;

2. investigating the concatenation of several FEC codes that may possibly improve performance. For instance, RSE is used in a first step to encode the content, and LDPC is then used in a second step to encode the symbols produced by RSE. Note that rateless codes such as Raptor and Online codes also rely on a two step encoding mechanism. This incites us to investigate other combinations of concatenations;

3. finding good unequal error protection (UEP) schemes. This can be an interesting solution when applied to multimedia streams (remember that I-frames are more important than P-frames). UEP was not satisfyingly solved in the past;

4. looking at cross-layer FEC protection with a single code.

This (non-exhaustive) list will certainly have to be refined and revised over time.
The latter aspect deserves some more explanations: Simulations carried out at DVB-H showed that in a well known environment, application layer FEC can favorably replace any FEC used at lower levels. More precisely, because the entire protocol stack is known, application layer FEC can be adapted to the behavior of each layer. This is a great advantage, since for instance, symbols could be aligned with layer-2 frames. This allows layer-2 erasure correction, while doing FEC encoding over the entire file, which is necessarily more efficient from an erasure recovery perspective. Other cross-layer optimizations that take into account for instance the channel loss pattern at layer-2 need to be invested.

15.2.2 Application Level Reliability

FLUTE/ALC does not offer a fully reliable distribution service when transmissions have finite duration (e.g. in *push* mode). The application level FEC mechanisms presented in this thesis largely improve the situation compared to a use case without any FEC, but this protection is not always sufficient.

File Repair Mechanisms

Therefore a complementary reliability mechanism, built on top of, or used in conjunction with FLUTE or ALC, is required in some cases. This has been done in MBMS with a dedicated file repair mechanism, whose scalability in case of widespread packet losses remains to be proven. Other repair mechanisms, e.g. peer-to-peer approaches that may reuse the FEC encoding of ALC or NORM, also need to be invested.

Using Peer-to-Peer in conjunction with FLUTE?

The peer-to-peer approach is especially of interest since, as we have mentioned before, these approaches increasingly rely on FEC. Besides peer-to-peer can be built on top of FLUTE (using FLUTE in unicast mode), as well as used in parallel using other transport protocols. It seems therefore quite straightforward to look how these approaches can be mapped to the reliable multicast use case.
Future work will address the problem of finding an optimal combination of these "application level" reliability techniques across a wide range of use cases. It could substantially improve large scale content distribution services. Ideally, any such scheme or optimization will improve scalability and independence from the underlying transport layer (i.e. ALC/FLUTE or NORM).

15.2.3 Security

Security is a subject that was out of scope of this thesis. However there is a strong need to gain experience in this area.
IPDC and MBMS provide security through a hybrid of DRM (Digital Rights Management) and specialized transport security protocols. The full advantage of IETF security mechanisms (such as IPsec and the building blocks defined by the MSEC (Multicast Security) working group) is yet to be explored and exploited. This is especially true in open networks (e.g. Wifi hot spots) and the public Internet.

TESLA

The Planète research team has started some activities in this area, in particular with an instantiation of the TESLA authentication building block to the specificities of ALC and NORM protocols, in order to provide a source authentication and message integrity service. This is only a first step towards a secure reliable multicast environment.

Fundamental Group Communication Security Schemes

More generally, group communications lack fundamental security schemes allowing to efficiently manage and implement authentication, encryption and group and content access rights in a scalable manner. Further research on this area is definitely needed.

Bibliography

[1] *Digital Fountain Inc.* URL: http://www.digitalfountain.com/.

[2] *M6Bone - IPv6 multicast network.* URL: http://www.m6bone.net/.

[3] *MPEG4IP project home page.* http://sourceforge.net/projects/mpeg4ip/.

[4] 3GPP. *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Setvice; Protocols and Codecs (Release 6.1.0)*, June 2005. 3GPP TS 26.346 v1.5.0.

[5] B. Adamson, C. Bormann, M. Handley, and J. Macker. *NACK-Oriented Reliable Multicast (NORM) Building Blocks*, Nov. 2004. Request for Comments 3941.

[6] B. Adamson, C. Bormann, M. Handley, and J. Macker. *Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol*, Nov. 2004. Request for Comments 3940.

[7] R. B. Adamson and J. Macker. Quantitative prediction of nack oriented reliable multicast (norm) feedback. In *Proc. IEEE MILCOM 2002*, Oct. 2002.

[8] H. Bai and M. Atiquzzaman. Error modeling schemes for fading channels in wireless communications: A survey. *IEEE Communications Surveys and Tutorials*, 5(2), 2003.

[9] S. Bajaj, L. Breslau, and S. Shenker. Uniform versus priority dropping for layered video. In *ACM SIGCOMM'98*, Sept. 1998.

[10] J. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the internet. In *ACM SIGCOMM'94*, Oct. 1994.

[11] C. Borman, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L.-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura, and H. Zheng. *RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed*, July 2001. Request for Comments 3095.

[12] A. Brown. Applications of algebraic-geometric codes over the erasure channel. In *IT-Lab Conference on Coding and Communications at ENST, Paris, France*, Nov. 2004.

[13] A. Burr. Turbo-codes: the ultimate error control codes? *Electronics and Communication Engineering Journal*, 2003.

[14] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. Flid-dl: Congestion control for layered multicast. In *2nd Workshop on Networked Group Communication (NGC2000)*, Nov. 2000.

[15] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *ACM SIGCOMM'98*, Aug. 1998.

[16] C. Castelluccia, G. Montenegro, J. Laganier, and C. Neumann. Hindering Eavesdropping via IPv6 Opportunistic Encryption. In *Proc. of* 9^{th} *European Symposium on Research in Computer Security (ESORICS 2004)*, volume 3193, pages 309–321. Lecture Notes in Computer Science (LNCS), Springer-Verlag, September 2004.

[17] A. Chaintreau, F. Baccelli, and C. Diot. Impact of network delay variation on multicast sessions performance with tcp-like congestion control. *IEEE Trans. on Networking (TON)*, pages 500–512, 2002.

[18] S. Cheung, M. Ammar, and X. Li. On the use of destination set grouping to improve fariness in multicast video distribution. In *IEEE INFOCOM'96*, Mar. 1996.

[19] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *IEEE SIGCOMM'90*, Sept. 1990.

[20] B. Classon and Y. Blankenship. *Low-Complesity LDPC coding for OFDMA PHY*, May 2004. Contribution to the IEEE 802.16 Broadband Wireless Access Working Group.

[21] L. Dairaine, L. Lancrica, and J. Lacan. Enhancing peer to peer parallel data access with peerfect. In *Fifth International Workshop on Networked Group Communication (NGC'03), Munich, Germany*, Sept. 2003.

[22] W. G. Davis. *SinisterSdp - An open source Session Description Protocol library.* URL: https://sourceforge.net/projects/sinistersdp/.

[23] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balenseifen. Deployment issues for the ip multicast service and architecture. *IEEE Networking magazine special issue on Multicasting*, 2000.

[24] H. Eriksson. Mbone: The multicast backbone. *Communications of the ACM*, 1994.

[25] H. Ernst, L. Sartorello, and S. Scalise. Transport layer coding for the land mobile satellite channel. In *59th IEEE Vehicular Technology Conference (VTC'04), Milan, Italy*, May 2004.

[26] ETSI. *IP Datacast Baseline Specification: Specification of Interface I MT (a080)*, Apr. 2004. DVB Interim Specification.

[27] S. Faurite, A. Francillon, and V. Roca. *TESLA source authentication in the ALC and NORM protocols*, July 2005. Work in Progress: <draft-faurite-rmt-tesla-for-alc-norm-00.txt>.

[28] G. D. Forney. The viterbi algorithm: A personal history. In *Viterbi Conference, University of Southern California, Los Angeles*, Mar. 2005.

[29] T. A. S. Foundation. *Xerces-C++ Parser.* URL: http://xml.apache.org/xerces-c/.

[30] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, Nov. 1999. Request for Comments 2046.

[31] R. G. Gallager. Low density parity check codes. *IEEE Transactions on Information Theory*, 8(1), Jan. 1962.

[32] J. Gemmell, E. Schooler, and J. Gray. Fcast multicast file distribution. *IEEE Network*, 14(1), Jan. 2000.

[33] M. Handley and V. Jacobson. *SDP: Session Description Protocol*, Apr. 1998. Request for Comments 2327.

[34] S. Kesha. Why cell phones will dominate the future internet. *ACM Computer Communication Review (CCR)*, 2005.

[35] A. Konrad, B. Zhao, A. Joseph, and R. Ludwig. A markov-based channel model algorithm for wireless networks. In *Proceedings of Fourth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2001.

[36] M. A. Kousa. A novel approach for evaluating the performance of spc product codes under erasure decoding. *IEEE Trans. Inform. Theory*, 50(1), Jan. 2002.

[37] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems Services*, Nov. 2000.

[38] J. E. Kurose and K. W. Ross. *Computer Networking - A Top-down Approach Featuring the Internet.* Addison-Wesley, Reading, Massachusetts, third edition, 2005.

[39] J. Lacan, L. Dairaine, L. Lancerica, and J. Fimes. Peerfect : Efficient peer to peer data access scheme using a fast integer-based cauchy erasure code. Research report, ENSICA, 2003.

[40] J. Lacan and J. Fimes. A construction of matrices with no singular square submatrices. In *7th International Conference on Finite Fields and Applications*, May 2003.

[41] E. Levinson. *The MIME Multipart/Related Content-type*, Aug. 1998. Request for Comments 2387.

[42] B. Li and J. Liu. Multirate video multicast over the internet: an overview. 17(1), Jan. 2003. IEEE Network, "Multicasting: an enabling technology".

[43] W. Li. Overview of fine granularity scalability in mpeg-4 video standard. 11(3), Mar. 2001. IEEE Transaction on Circuits and Systems for Video Technology.

[44] M. Luby. Lt codes. In *43rd IEEE Symposium on Foundations in Computer Science*, Nov. 2002.

[45] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. *Asynchronous Layered Coding (ALC) protocol instantiation*, Dec. 2002. Request for Comment 3450.

[46] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, M. Handley, and J. Crowcroft. *Forward Error Correction Building Block*, Dec. 2002. Request for Comments 3452.

[47] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, M. Handley, and J. Crowcroft. *Layered Coding Transport (LCT) building block*, Dec. 2002. Request for Comments 3451.

[48] M. Luby and V. Goyal. *Wave and Equation Based Rate Control Building Block*, Apr. 2004. Request for Comments 3738.

[49] M. Luby, V. Goyal, S. Skaria, and G. Horn. Wave and equation based rate control using multicast round trip time. In *ACM SIGCOMM'02*, Aug. 2002.

[50] M. Luby, J.Gemmell, L.Vicisano, L.Rizzo, J. Crowcroft, and B. Lueckenhoff. *synchronous Layered Coding: A massively scalable reliable multicast protocol*, Jan. 2000. Work in Progress: <draft-ietf-rmt-pi-alc-01.txt>.

[51] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Improved low-density codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2), Feb. 2001.

[52] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. 29th Symp. on Theory of Computing*, pages 150–159, May 1997.

[53] M. Luby, A. Shokrollahi, V. Stemann, M. Mitzenmacher, and D. Spielman. *Irregularly graphed encoding technique*, June 2000. U.S. Patent No. 6,081,909.

[54] M. Luby, A. Shokrollahi, V. Stemann, M. Mitzenmacher, and D. Spielman. *Loss resilient decoding technique*, June 2000. U.S. Patent No. 6,073,250.

[55] M. Luby, A. Shokrollahi, V. Stemann, M. Mitzenmacher, and D. Spielman. *Message encoding with irregular graphing*, Dec. 2000. U.S. Patent No. 6,163,870.

[56] M. Luby, A. Shokrollahi, V. Stemann, M. Mitzenmacher, and D. Spielman. *Loss resilient code with double heavy tailed series of redundant layers*, Feb. 2001. U.S. Patent No. 6,195,777.

[57] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. *Raptor Forward Error Correction Scheme*, Sept. 2005. Work in Progress: <draft-ietf-rmt-bb-fec-raptor-object-02.txt>.

[58] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. *The use of Forward Error Correction (FEC) in reliable multicast*, Dec. 2002. Request for Comments 3453.

[59] M. Luby, L. Vicisano, and A. Haken. *Reliable multicast transport building block: Layered Congestion Control*, Nov. 2000. Work in Progress: <draft-ietf-rmt-bb-lcc-00.txt>.

[60] J. O. M. Nilsson, D. Dalby. Layered audiovisual coding for multicast distribution on ip networks. In *6th IEEE, European Workshop on Distributed Imaging*, Nov. 1999.

[61] D. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, ISBN: 0521642981, 2003.

[62] P. Maymounkov. Online codes. Research Report TR2002-833, New York University, Nov. 2002.

[63] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[64] P. Maymounkov and D. Mazires. A construction of matrices with no singular square submatrices. In *Proc. of IPTPS'03*, Feb. 2003.

[65] H. Mehta, R. Walsh, I. Curcio, J. Peltotalo, and S. Peltotalo. *SDP Descriptors for FLUTE*, May 2005. Work in Progress: <draft-ietf-rmt-flute-sdp-02.txt>.

[66] M. Murata, S. S. Laurent, and D. Kohn. *XML Media Types*, Jan. 2001. Request for Comments 3023.

[67] R. Neal. *Software for Low Density Parity Check (LDPC) codes.* http://www.cs.toronto.edu/∼radford/ldpc.software.html.

[68] C. Neumann, J. Laganier, C. Castelluccia, and G. Montenegro. *An Open Source Implementation of Opportunistic Encryption.* URL: http://www.inrialpes.fr/planete/people/chneuman/OE.html.

[69] C. Neumann and V. Roca. Multicast streaming of hierarchical mpeg4 presentations. In *ACM Multimedia 2002 (Short Paper), Juan les Pins, France*, Dec. 2002.

[70] C. Neumann and V. Roca. Scalable video streaming over alc (svsoa): a solution for the large scale multicast distribution of videos. Research Report 4769, INRIA, Mar. 2003.

[71] C. Neumann and V. Roca. *A Survey of Large Scale Multimedia Streaming Techniques for the Internet*, Nov. 2003. Tutorial presented during the International Workshop on Multimedia Interactive Protocols and Systems (MIPS'03), Napoli, Italy.

[72] C. Neumann and V. Roca. Analysis of fec codes for partially reliable media broadcasting schemes. In *2nd ACM International Workshop on Multimedia Interactive Protocols and Systems (MIPS'04), Grenoble, France*, Nov. 2004.

[73] C. Neumann and V. Roca. Scalable video streaming over alc (svsoa): a solution for the large scale multicast distribution of videos. In *Streaming media distribution over the Internet (SMDI04), Athens, Greece*, May 2004.

[74] C. Neumann and V. Roca. Impacts of the startup behavior of multilayered multicast congestion control protocols on the performance of content delivery protocols. In *IEEE 10th International Workshop on Web Content Caching and Distribution (WCW 2005), Sophia Antipolis, French Riveria, France*, Sept. 2005.

[75] C. Neumann, V. Roca, A. Francillon, and D. Furodet. Impacts of packet scheduling and packet loss distribution on fec performances: Observations and recommendations. In *ACM CoNEXT'05 Conference, Toulouse, France*, Oct. 2005.

[76] C. Neumann, V. Roca, A. Francillon, and D. Furodet. Impacts of packet scheduling and packet loss distribution on fec performances: Observations and recommendations. Research Report 5578, INRIA, May 2005.

[77] C. Neumann, V. Roca, J. Labouré, and Z. Khallouf. *An Open-Source Implementation of a Low Density Parity Check (LDPC) Large Block FEC Code.* URL: http://www.inrialpes.fr/planete/people/roca/mcl/.

[78] C. Neumann, V. Roca, and R. Walsh. *A File Aggregation Scheme for FLUTE*, June 2005. Work in Progress: <draft-neumann-rmt-flute-file-aggregation-01.txt>.

[79] C. Neumann, V. Roca, and R. Walsh. Large scale content disrtibution protocols. *ACM Computer Communication Review (CCR)*, 35(5), Oct. 2005.

[80] Y. Nomura, R. Walsh, J.-P. Luoma, H. Aseada, and H. Schulzrinne. *A Framework for the Usage of Internet Media Guides*, July 2005. Work in Progress: <draft-ietf-mmusic-img-framework-08.txt>.

[81] T. Paila. Mobile internet over ip data broadcast. In *10th IEEE Int. Conference on Telecommunications (ICT'03), Papeete, French Polynesia*, Jan. 2003.

[82] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. *FLUTE - File Delivery over Unidirectional Transport*, Oct. 2004. Request for Comments 3926.

[83] J. Peltotalo and S. Peltotalo. *MAD/TUT Project: an Open Source Implementation of FLUTE.* URL: http://www.atm.tut.fi/mad/.

[84] S. Peltotalo, J. Peltotalo, and V. Roca. *Simple XOR, Reed-Solomon, and Parity Check Matrix-based FEC Schemes*, June 2004. Work in Progress: <draft-peltotalo-rmt-bb-fec-supp-xor-pcm-rs-00.txt>.

[85] A. Perrig. *TESLA Implementation.* URL: http://www.ece.cmu.edu/ adrian/tesla.

[86] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS '01*, pages 35–46, Feb. 2001.

[87] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5(Summer), 2002.

[88] J. Plank and M. Thomason. On the practical use of ldpc erasure codes for distributed storage applications. Research Report UT-CS-03-510, University of Tennessee, Sept. 2003.

[89] J. S. Plank, A. L. Buchsbaum, R. L. Collins, and M. G. Thomason. Small parity-check erasure codes - exploration and observations. In *DSN-05: International Conference on Dependable Systems and Networks*, June 2005.

[90] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the oceanstore prototype. In *2nd USENIX Conference on File and Storage Technologies (FAST 2003)*, Mar. 2003.

[91] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), Apr. 1997.

[92] L. Rizzo. pgmcc: a tcp-friendly single-rate multicast congestion control scheme. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 17–28, New York, NY, USA, 2000. ACM Press.

[93] V. Roca and al. *MCLv3: an Open Source GNU/GPL Implementation of the ALC and NORM Reliable Multicast Protocols.* URL: http://www.inrialpes.fr/planete/people/roca/mcl/.

[94] V. Roca, Z. Khallouf, and J. Laboure. Design and evaluation of a low density generator matrix (ldgm) large block fec codec. In *Fifth International Workshop on Networked Group Communication (NGC'03), Munich, Germany*, Sept. 2003.

[95] V. Roca and B. Mordelet. Improving the efficiency of a multicast file transfer tool based on alc. In *IEEE International Symposium on Computer Communications (ISCC'02)*, July 2002.

[96] V. Roca and B. Mordelet. Improving the efficiency of a multicast file transfer tool based on alc. Research Report 4411, INRIA, Mar. 2002.

[97] V. Roca and C. Neumann. Design, evaluation and comparision of four large block fec codes, ldpc, ldgm, ldgm staircase and ldgm triangle, plus a reed-solomon small block fec codec. Research Report 5225, INRIA, June 2004.

[98] V. Roca and C. Neumann. *Low Density Parity Check (LDPC) Forward Error Correction*, June 2005. Work in Progress: <draft-roca-rmt-ldpc-00.txt>.

[99] S. Servetto, R. Puri, J.-P. Wagner, P. Scholtes, and M. Vetterli. Video multicast in (large) local area networks. In *IEEE INFOCOM'02*, June 2002.

[100] A. Shokrollahi. Raptor codes. Research Report DR2003-06-001, Digital Fountain, 2003.

[101] L. Vicisano, L. Rizzo, and J. Crowcroft. Tcp-like congestion control for layered multicast data transfer. In *IEEE INFOCOM'98*, Feb. 1998.

[102] B. Vickers, C. Albuquerque, and T. Suda. Source adaptive multi-layered multicast algorithms for real-time video distribution. 8(6), Dec. 2000. IEEE Transaction on Networking.

[103] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum algorithm. *IEEE Trans. Inform. Theory*, IT-13, Apr. 1967.

[104] B. Watson, M. Luby, and L. Vicisano. *Forward Error Correction (FEC) Building Block*, Sept. 2005. Work in Progress: <draft-ietf-rmt-fec-bb-revised-01.txt>.

[105] T. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby. *Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer*, Jan. 2001. Request for Comments 3048.

[106] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 275–285, New York, NY, USA, 2001. ACM Press.

[107] M. Yajnik, S. B. Moon, J. F. Kurose, and D. F. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *IEEE INFOCOM'99*, pages 345–352, 1999.