OPTIMIZATION OF RESOURCES IN A DIGITAL CINEMA NETWORK

A Degree's Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Pedro Lladó González

In partial fulfilment
of the requirements for the degree in
AUDIOVISUAL SYSTEMS ENGINEERING

Advisor: Jorge Mata

Barcelona, February 2014
Abstract

This project is a study of a common digital cinema network, focusing on sending big data files, which are films, and trying to improve the performance of the delivery of these data sending operations from the distributor to the cinemas.

First of all, an analysis of the basic structures of the digital cinemas is made, from the servers and projectors in a booth to the bases of the Digital Cinema Initiatives, explaining how cinema packages should be made and how cinema exhibitors were allowed to afford the digitalisation.

Afterwards, the way the current platform works has been studied, its weaknesses and its improvable problems, noting on the available protocols and the way the current platform use them to achieve the deliveries.

Then, when all the necessary information is collected, a plot of purposes is set trying to improve the behaviour of the platform.
Resum

Aquest projecte és un estudi d'una xarxa de cinema digital típica, centrant-nos en l'enviament d'arxius de gran tamany, les pel·lícules, i intentant millorar el comportament d'aquests enviaments de dades des de la distribuidora fins als cinemes.

Per començar, s'ha fet una anàlisi de les estructures bàsiques del cinema digital, des dels servidor i els projectors en una sala fins a les bases del DCI (“Digital Cinema Initiatives”), explicant com s'han de fer els paquets de pel·lícules i com els exhibidors han pogut assolir la digitalització.

Tot seguit, s'ha estudiat el funcionament de una plataforma actual per a veure com funciona, les seves debilitats i els seus problemes a millorar, tenint en compte els protocols que ja existeixen i la manera en la que la plataforma assoleix les entregues.

Després de la recopilació de la informació necessària, s'ha fet una sèrie de propostes per a millorar el comportament de la plataforma.
Resumen

Este proyecto es un estudio de una red de cine digital típica, centrándose en el envío de archivos de gran tamaño, las películas, e intentando mejorar el comportamiento de estos envíos de datos desde la distribuidora hasta los cines.

Para empezar se ha hecho un análisis de las estructuras básicas del cine digital, desde los servidores y los proyectores de una sala hasta las bases del DCI (“Digital Cinema Iniciatives”), explicando como se tienen que organizar estos paquetes de películas y como los exhibidores han podido asumir la digitalización.

Para continuar, se ha estudiado el funcionamiento de una plataforma actual para ver como funciona, sus debilidades y sus problemas a mejorar, teniendo en cuenta los protocolos existentes y la forma en la que la plataforma consigue realizar las entregas.

Tras la recopilación de la información necesaria se han hecho una serie de propuestas para mejorar el comportamiento de la plataforma.
Acknowledgements

When I started this thesis I didn't know much about digital cinema nor satellite communications. Many parameters of the platform, its protocols and the way it works has been very difficult to understand because the original platform was created by an engineers team that is not working on it nowadays; getting this information has been tough and a slow process.

About communications I had some base learnt during my degree but I had to read a lot, discover new things and my tutor Jorge Mata helped a lot at providing some useful information to structure the project and advising about protocols to be studied.

Furthermore, I have had to read a lot about everyone of the procedures of the project, being so useful for the self learning and realizing that it's not that easy to write about a project.
# Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02/01/2015</td>
<td>Document creation</td>
</tr>
<tr>
<td>1</td>
<td>24/01/2015</td>
<td>Document revision</td>
</tr>
<tr>
<td>2</td>
<td>29/04/2015</td>
<td>Document revision</td>
</tr>
<tr>
<td>3</td>
<td>13/05/2015</td>
<td>Document revision</td>
</tr>
<tr>
<td>4</td>
<td>15/05/2015</td>
<td>Document delivery</td>
</tr>
</tbody>
</table>

## DOCUMENT DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedro Lladó</td>
<td><a href="mailto:lladopedro@gmail.com">lladopedro@gmail.com</a></td>
</tr>
<tr>
<td>Jorge Mata</td>
<td><a href="mailto:jmata@entel.upc.edu">jmata@entel.upc.edu</a></td>
</tr>
</tbody>
</table>

Written by:  
Date 15/05/2015  
Reviewed and approved by:  
Date 15/05/2015  
Name Pedro Lladó  
Position Project Author  
Name Jorge Mata  
Position Project Supervisor
# Table of contents

The table of contents must be detailed. Each chapter and main section in the thesis must be listed in the "Table of Contents" and each must be given a page number for the location of a particular text.

## Index

Abstract.................................................................................................................................2  
Resum.................................................................................................................................3  
Resumen...............................................................................................................................4  
Acknowledgements..............................................................................................................5  
Revision history and approval record..................................................................................6  
Table of contents..................................................................................................................7  
List of Figures.......................................................................................................................9  
1. Introduction....................................................................................................................10  
   1.1. Digital Cinema Package (DCP)................................................................................10  
      1.1.1. History..............................................................................................................10  
      1.1.2. DCP Structure.................................................................................................10  
   1.2. Projectors and servers.............................................................................................12  
   1.3. Virtual Print Fee ....................................................................................................12  
2. State of the art................................................................................................................14  
   2.1. Data sending infrastructures....................................................................................14  
      2.1.1. Courier............................................................................................................14  
      2.1.2. DSL and VPN.................................................................................................14  
      2.1.3. Satellite...........................................................................................................15  
   2.2. Current Protocols.....................................................................................................15  
      2.2.1. NORM.............................................................................................................16  
      2.2.2. FEC................................................................................................................17  
      2.2.3. PGM.................................................................................................................18  
   2.3. Related Projects......................................................................................................20  
      2.3.1. FLUTE.............................................................................................................20  
      2.3.2. Spread Toolkit................................................................................................21  
      2.3.3. The JGroups Project.......................................................................................21  
      2.3.4. OpenPGM.......................................................................................................22  
      2.3.5. UDPcast.........................................................................................................22
2.4. Current Tools

2.4.1. OpenVPN

2.4.2. Satellite

2.4.3. RBC Protocol

2.5. Platform Overview

3. Project development

3.1. Scenario

3.2. Performance of current platform

3.2.1. Lossless Transmission

3.2.2. Analysis of the losses

3.3. Current Platform Problematic

3.4. Improvements proposal

3.4.1. Topology improvements proposal

3.4.2. Protocol improvements proposal

4. Results

4.1. Results by applying topology improvements

4.2. Results by applying protocol improvements

5. Budget

6. Environment Impact

7. Conclusions and future development

Bibliography

Appendix A: DCP Example

Appendix B: RBC Protocol

Glossary
List of Figures

- figure 1: Scenario of the platform (page 28)
- figure 2: Unencrypted TCP packet (page 30)
- figure 3: Encrypted UDP packet on the VPN (page 30)
- figure 4: Lossless transmission from buffer to sat hub (page 31)
- Figure 5: Lossless satellite transmission (page 33)
- figure 6: Analysis of an error in the transmission from buffer to sat hub (page 34)
- figure 7: topology modified scenario with regions (page 37)
- figure 8a: explanation of regions distribution (page 37)
- figure 8b: explanation of regions distribution (page 38)
- figure 9: RBC Header (page 50)
- figure 10: different values of type field in RBC header (page 50)
- figure 11: Stream Desc breakdown (page 51)
- figure 12: Stream Data breakdown (page 52)
- figure 13: Stream Key breakdown (page 53)
- figure 14: Stream Checkpoint breakdown (page 54)
- figure 15: Stream Status breakdown (page 55)
- figure 16: Stream Bandwidth breakdown (page 56)
- figure 17: Stream Switch breakdown (page 57)
1. **Introduction**

1.1. **Digital Cinema Package (DCP)**

1.1.1. **History**

After many years using 35mm reels, distribution companies realized that creating digital copies of the films they distribute would be immensely cheaper. It is due to the cost of every copy in the non-digital format, because of the reels, the negatives, the chemical products, the transport of the reels, the amount of people you need to make it possible...

Digital Cinema Initiatives (DCI), a joint venture of the six major studios, created a standard about how DCP (Digital Cinema Package) must be done in order to fit the purposes of quality and compatibility with digital equipment. DCP is the new format of the film. It's a folder that contains all the necessary information to play this film.

They had the responsibility of solving a great problem, which would be the security. Copying, distributing and playing digital content would be much easier than with reels, and make it illegally would be easier too, so they created the Key Delivery Messages (KDM) to encrypt the digital content too. A KDM is a key that only opens one film in one cinema server, and during a period of time, which is supposed that is highly reliable.

These multinational distributors thought that all the theatres needed to digitalize themselves because it was extremely efficient for them, but theatre managers didn't understand that, just because they didn't win anything: they were ready to work with analogical reels and they couldn't afford the inversion of the digitalization. For a theatre, the digitalization meant about 50,000€ per booth, and they knew that if they had rebounded this to the clients, they had lost a lot of money, most of all, because the final service would be near exactly the same: a high quality film played in a cinema booth.

Digital theatres uses a server in each booth. These servers are the ones who allow to store the content and play it through the projector, so they are connected via booth LAN. Moreover, all the booths are connected between them, and with a Theatre Central Server (TCS). It means that due to the theatre network, all the content can be easily managed from the TCS, can be distributed and programmed too, sharing playlists between different booths, only possible if this network exists.

1.1.2. **Structure**

A complete DCP is formed by the following files:

- **CPL (Composition Playlist)**
  Is a text file which lists video, sound, and subtitle (if needed) files which the server looks for when a film is being screened. Since images, sound and subtitles are divided into several files in a DCP (which are referred to as digital reels), the CPL lists these reels in the order need to be played and when these reels need to be played. It's the pattern which guide the DCP into a correct screening order and timing.
A DCP can contain more than a CPL. For example, if two versions of the same film shares many of their files, they can use the same DCP (folder) and two CPLs.

**PKL (Packing List)**
Is a text file which lists all the files of the DCP. If the DCP contains two versions of the film, the PKL will contain two CPL and for example, the whole audio of two different languages. It helps to realize if the DCP is not complete.

**Assetmap**
An Assetmap maps files to numbers. Numbers are used extensively in digital cinema packages to identify things. Everything (picture, video and audio files, subtitle files, compositions etc.) has a UUID (Universally Unique Identifier). When ingesting a digital cinema package these number-file pairs will be added to an internal dictionary which keeps track of all the numbers and files. The mechanism will reproduce the essential property of the Assetmap: Numbers pointing at files. Every time a playback system needs something for playback it will search the internal dictionary for the required number. The dictionary will know where to find the asset with that number.

**Volindex**
A single DCP may be stored in more than one medium (e.g., multiple hard disks). The xml file VOLINDEX is used to identify the volume order in the series. That's why almost all the DCP has the same Volindex, because usually the whole DCP it's located in the same medium.

**Video and Audio files**
A DCP usually contains many video and audio files. That's why the whole film is composed by “reels” due to maintain the same composition as in 35mm. Apart of that, it makes easy work with them than with a high size file at the postproduction, and every step before its reproduction.

The DCP usually has as many audio files as video. That's why it's easier to mix one video reel with an audio reel than being composing different duration files so they fit in number and duration.

In case of multilingual features, separate audio reels are required to convey different languages.

This video and audio files are contained in a .mxf. Those mxf are wrappers (containers) that can contain many different kind of files, as video or audio in this case. The whole video of a film has usually a size of centenars of GB. Audio and video reels usually are about 20 minutes of a film (according to 35mm reels), and it means about 20-30GB per reel in video and about 2GB per audio reel.
Subtitle files

The subtitle files are organised in folders which contains a .ttf and a .xml. The subtitles files are text files, only plain text and font archive, so their size is about kB.

*An example of a complete DCP is in the annex A.

1.2. Projectors and Servers

To play digital cinema content, not only a projector is needed. A server to locate the digital films is needed. This digital servers usually has a couple TB of capacity. They have an interface to be able to manage the contents, program the sessions and establish communication with the projector, which is of course necessary to reproduce.

The projector only converts the information of the server into light, through the media block. DLP (Digital Light Processing) displays are used to convert white channel to the three channels of light (R,G,B), using three DMDs (Digital Micromirror Device). The complexity of the projectors is about the amount of light emitted, the calibration of the channels and the alignment of the DMDs (if they are not aligned, a white stripe would be just a red, green and blue stripes). The refrigeration of the projector is a very important factor to take into account because the amount of power needed is very high and the temperature of the lamp and the electronic components need much more cooling than simple fans.

These equipments are connected to the cinema network to be able of being managed from the TCS or from the office.

1.3. Virtual Print Fee

In traditional cinema business, content providers were responsible of the print costs. In the digital world, these costs have been reduced significantly (about 90%). VPF was created as a model to match exhibitors costs and distributors savings.

A Virtual Print Fee is a financing mechanism for funding the purchase of digital cinema equipment. It is structured to cover the costs of converting to digital projection equipment through a fee that distributors pay for each booking over a set period of time. The idea behind a VPF is that the distributors save money by shipping digital, rather than 35mm film prints, and these savings are used to contribute to the cost of equipment for exhibitors.

First model of VPF, in USA, worked well because some enterprises took the risk of the initial invest, the cost of the equipment was discussed with the manufacturers and big companies, which assumed a part of it with their savings obtained by using this technology. This model, standardized by the DCI, has a deadline.

In each country, VPF has its own parameters and conditions. For example, in Spain, VPF contracts can’t be established any more, and when the current VPF contracts end, each exhibitor will have to buy its equipment.

A common model in Spain is that the exhibitor buy the equipments selecting its best financial solution (bank loan). Then, the install enterprise receive money from distributors
every time a film is shown, and an amount of this money is paid to the exhibitor, until it covers a high amount of the price of the equipment. After these years of paying the equipment from the installator, the equipment is in propriety of the installator.
2. **State of the art of the technology used or applied in this thesis:**

2.1. **Data sending infrastructures**

2.1.1. Courier

Nowadays, many DCP are sent to the cinemas on Hard Disk (HD) format. That is creating a DCP and burning it into a common HD according to DCI. They are sent by common transport enterprises as a standard packet. It doesn't make sense if they can be delivered in a non-physical format with the current technology. The trend is trying to avoid this method and looking for cheaper and efficient ways.

2.1.2. DSL and VPN (Digital Subscriber Line and Virtual Private Network)

A DCP contains common types of files that are widely sent by regular DSL using common transfer protocols. These DCP fit perfectly to any of these protocols, so DSL is a very cheap, easy and convenient way to transfer cinema files.

But for secure communications, enterprises need to use private networks, and when the geographical situation is not located in a concentrated place, they need to use a way to simulate it and work as if they were in the same work network.

In the past, they have used leased lines over long distances, but nowadays, Virtual Private Networks (VPN) works pretty well. VPNs were created to achieve using a work network using the Internet to facilitate communications. Internet access is cheap, but it's not secure enough to run private information, so VPNs are designed to create sure, encrypted Internet Protocol (IP) tunnels to communicate between geographically-distant networks across the Internet.

Tunnelling is a method which data is transferred across a network between two endpoints. VPNs use tunnels to establish end-to-end connectivity. A packet destined to a remote network is encapsulated by adding additional header and then, it's sent across the network to the endpoint. The header is removed there and the packet is sent out onto the remote network.

VPN offers benefits as:
- flexibility: a user can connect to the remote network from any place with Internet connection.
- transparency: through tunnelling which allows arbitrary traffic to traverse the VPN.
- security: by using encryption and authentication. Encryption provides privacy by scrambling the data in the tunnel.
- low cost: the cost of a VPN is much less than the cost of using dedicated lines, particularly by using freely available open source VPN.
But:

VPN solutions are usually deployed to provide access over the Internet which sometimes varies in the availability and bandwidth of the connection, so we depend on the Internet operator to be able to use the VPN.

2.1.3. Satellite

Realizing that we need to send the same DCP to many different cinemas, seems to make sense to use multicasting to achieve it. Multicasting allows us to allocate all the necessary clients and the clients could inform about their state through a back channel mechanism (usually ACK or NAK).

A really important system to take into account if we need to send via multicast is using a satellite transmissions. They offer multicasting, which is reliable enough and its bandwidth should be wide enough. It means an expensive cost, but this cost is fixed (no matter its usage) and if the platform is going to be transmitting enough it will be highly effective, and the transmission cost will be cheap ultimately. Probably it would be cheaper it would use regular DSL or Optical Fiber, but the problem is that it would be too expensive if a good infrastructure is not set, and it's the case, because depending on the geographical location of the cinema, its internet is very poor.

To achieve the satellite multicast requirements, DVB protocols fit perfectly with the platform necessities. Digital Video Broadcasting is a worldwide organization dedicated to develop open standards for digital TV. It was created by the union of many private enterprises that were looking for standardize the digital television progresses.

Since then, a lot of protocols have been created to improve the performance of the video broadcasting infrastructure, as for example:

- DVB-S → To run video broadcasting through satellite.
- DVB-C → To send digital video content through cable.
- DVB-T → For the Terrestrial Digital Broadcasting TV system
- DVB-H → Included in DVB-T, specifying the transmission for portable devices (hand held)

Other standards have been created by the DVB organization but with less impact than these ones.

2.2. Current Protocols

To face this situation, protocols used in this area will need to stand reliability and multicasting: Reliable Multicast Protocols.

Multicast, by its nature, is not a connection-oriented mechanism, so traditional protocols which allows retransmissions of missing packets are not appropriate. If we are dealing applications as streaming, the dropped packets are not a problem, but for distribution of critical data, a mechanism is required for requesting retransmission. As far as we need to be completely reliable (no matter if we need retransmission), we need to find out some protocols that allows a trustworthy multicast.
Two ways to achieve the reliable multicast purpose that fit perfectly on our platform are: Nack-Oriented Reliable Multicast (NORM) and Forward Error Correction (FEC).

2.2.1. NORM (Nack-Oriented Reliable Multicast, IETF RFC 5740)

This protocol can provide end-to-end reliable transport of bulk data objects or streams over generic IP multicast routing and forwarding services. NORM (NACK-Oriented Reliable Multicast) uses a selective, negative acknowledgement mechanism for transport reliability and offers additional protocol mechanisms to allow operations with a minimal a priori coordination among senders and receivers.

NORM can provide reliable transport of data from one or more senders to a group of receivers over an IP multicast network. The primary design goals of NORM are to provide efficient, scalable, and robust bulk data (e.g., computer files, transmission of persistent data) transfer across possibly heterogeneous IP networks and topologies. NORM allows senders and receivers to dynamically join and leave multicast sessions at will with minimal overhead for control information and timing synchronization among participants. To accommodate its capability, NORM protocol message headers contain some common information allowing receivers to easily synchronize to senders throughout the lifetime of a reliable multicast session. NORM is self-adapting to a wide range of dynamic network conditions with little or no pre-configuration. The protocol is tolerant of inaccurate timing estimations or lossy conditions that can occur in many networks. The protocol can also converge and maintain efficient operation even in situations of heavy packet loss and large queuing or transmission delays.

The NORM protocol design is principally driven by the assumption of a single sender transmitting bulk data content to a group of receivers. It is anticipated that multiple senders are allowed, but they will transmit independently of each another and receivers will maintain state as necessary for each sender. NORM identifies transmitted content (Norm-Objects) with transport identifiers that are applicable only while the sender is transmitting the given object. These transport data content identifiers (Transport Ids) are assigned in a monotonically increasing fashion by each NORM sender during the course of a NORM-Session. Participants, including senders, in NORM protocol sessions are also identified with unique identifiers (UUID). Each sender maintains its Transport Id assignments independently and thus individual Norm-Objects can be uniquely identified during transport by concatenation of the session-unique sender identifier (NormNodeId) and the assigned NormTransportId. The NORM protocol provides mechanisms so the sender application can terminate transmission of data content and inform the group of this in an efficient manner.

Senders transmit data content to the multicast session. Sender will transmit data content at a rate, with messages of size and network requirements. When congestion control mechanisms are needed, the sender transmission rate shall be controlled by the congestion control mechanism (it is recommended all data transmissions from multicast senders to be limited by the application congestion or congestion control algorithm). It's expected that there will be overlap and multiplexing of new data content transmission with repair content, due to the good utilization of the available capacity.

Apart of data content, the other messages may be employed as part of the protocol operation. Reliability of these protocol messages may be attempted by redundant
transmission when positive acknowledgement is prohibitive. Each receiver will respond with NACKs for any outstanding repairs they require and the sender should allow sufficient time between redundant transmission, in order to receive any NACK responses from the receivers.

The NACK messages generated includes information detailing their current repair needs. The identification of repair needs is dependent upon the data content identification. For the indicated transport entity, the NACK content will the identify the specific needed blocks to reconstruct the complete transmitted data.

The format of NACK content will depend on the data service model and the format of data content identification the protocol uses. It is recommended that transport data content identification is done within the context of a sender in a given session.

The format of NACK messages should enable the following:
- Transport data units required to repair the received content.
- Simple processing for NACK aggregation and suppression.
- Inclusion of NACKs for multiple objects in a single message
- A reasonably compact format.

### 2.2.2. FEC (Forwarding Error Correction, IETF RFC 3454)

Forward Error Correction codes provide a reliability method that can be used to augment or replace other reliability methods, specially for one-to-many reliability protocols such as reliable IP multicast. We first briefly review some of the basic properties and types of FEC codes before reviewing their uses in the context of reliable IP multicast.

The primary application of FEC codes to IP multicast protocols is as an erasure code. The payloads are generated and processed using an FEC erasure encoder and objects are reassembled from reception of packets containing the generated encoding using the corresponding FEC erasure decoder. The FEC encoder generates some number of encoding symbols that are of the same length as the source symbols and these encoded symbols are placed into each packet. Moreover, in each packet is placed enough information to identify the particular encoding symbols into the corresponding FEC decoder to recreate an exact copy of the source symbols. Ideally, the FEC decoder can recreate an exact copy from any of the encoding symbols.

The input to a block FEC encoder is k source symbols and n encoding symbols. The encoder generates n-k redundant symbols yielding an encoding block of n encoding symbols in total composed of the k source symbols.

A block FEC decoder has the property that with any k of the n encoded symbols is sufficient to reconstruct the original k symbols. For practical FEC codes, slightly more than k encoding symbols are needed.

It's said that FEC works averaging the noise. That's why, once we know the performance of the channel and we know the % of wrong bits in the receiver, the redundance allows the receiver to obtain only the needed ones. When a system uses FEC, it tends to work well above a certain minimum signal-to-noise ratio and not below it.
The main categories of FEC are “Block codes” and “Convolutional codes”. Block codes have a predetermined size. These blocks usually can be decoded to their block length using polynomial time algorithms (Time is bounded by a polynomial expression in the size of the input for some constant k). Convolutional codes have no predetermined size so they are more difficult to decode. Most algorithm to decode is the Viterbi algorithm (is a very complex algorithm used to find the most likely sequence of hidden states).

Very often, different block codes are concatenated to optimize the coding, but it increases the difficult of the decoding too. It’s usual to use convolutional code to be better coded, and then a large block code wrap it to avoid any error made by the convolutional codification.

Turbo codes are commonly used too and they consist in two or more convolutional codes and an interleaver to produce a block code. These are high-performance FEC, practically close to channel capacity limit. This code is very near to Shannon Limit (Noisy-channel coding theorem; establishes that for any given degree of noise in a a communication channel, there is nearly error-free maximum rate for discrete data. This theoretical maximum of transfer rate for a particular noise is the Shannon Limit).

Low-Density Parity-Check are efficiently linear block codes made from many single parity check codes. Their performance are very good and they fit very well on the channel capacity using an iterated soft-decision decoding approach. These codes are now used in many recent high-speed communication standards.

The performance of these codes depends on its code structure, which is made up of code rate, constraint length, block size, interleaving pattern and number of decoding iterations.

### 2.2.3. PGM (Pragmatic General Multicast, IETF RFC 3208)

PGM is a reliable multicast transport protocol for applications that need duplicate-free multicast data (orderer or unordered) delivery from multiple sources to multiple receivers. This protocol guarantees that every receiver receives correctly all data or if not, is able to detect unrecoverable packet loss. It’s thought to be a real solution for multicast applications with basic reliability requirements, focusing on its simplicity, scalability and network efficiency. It runs over datagram multicast protocol such as IP multicast and receivers give feedback with NAKs when detect that any sequence is missing. PGM defines a procedure for a reliable NAK forwarding to minimize NAK losses. To do that, receiver sends NAKs until sender send NCF (NAK Confirmation).

PGM tries to avoid the main reasons of being inefficient which are implosion of NAK and NCF, repair latency from the source and the propagation of disinterested receivers.

Source Path Messages (SPM) function stablish source path state establish source path state for a given Transport Session Identifier (TSI) in all PGM network elements on the distribution tree from the source. This information is used to address returning unicast and insure that NAKs return from the receiver on the reverse path.

As an end-to-end transport protocol, PGM specifies the formats of the packets and its procedures to transmit from a source and for receivers to receive data. This protocol specifies some elements to improve the reliability of NAKs to enhance the efficiency of the data transfers:
The source specifies information to the group about: Original data packets, the source paths to the group (SPM), multicast NCFs (in response to every NAK received) and repaired data packets (in response to NAKs too).

The receiver: use SPMs to send their NAKs, receive the original data and eliminate any duplicate packet, transmit repeatedly each NAK until it receives a matching NCF and gets ready to receive any data packet even if it's waiting a previous repaired packet.

Network elements: intercept and use the SPMs, multicast NCFs to the group (records TSI, the sequence number of the NAK and the input interface in which the NAK was received), forward repeatedly the first copy of a NAK until it gets the NCF, discard any duplicated repaired NAK and forward with the matching NCF and forward the repaired data packets on the NAK receiving interface.
2.3. Related Projects

2.3.1. FLUTE (File Delivery over Unidirectional Transport, IETF RFC 3926)

“FLUTE is a protocol for the unidirectional delivery of files over the Internet, which is particularly suited to multicast networks. The specification builds on Asynchronous Layered Coding (ALC), the base protocol designed for massively scalable multicast distribution.”

FLUTE is designed to unidirectional deliveries, so the figure of Sender cannot receive files. Files are delivered as transport objects and allows send encoded contents, as zipped files. It supports IPv4 and IPv6 and it's available sending to a large number of receivers. Protocol is thought to unidirectional deliveries but receivers can send information messages to the sender, in order to have feedback about the state of the transmission. The way to achieve reliability is using the Forward Error Correction and retransmissions.

It particularly suits to multicast network, supporting Any Source Multicast (ASM) and Specific Source Multicast (SSM), and can be used with both multicast and unicast UDP (User Datagram Protocol).

FLUTE builds on Asynchronous Layered Coding protocol instantiation of the Layered Coding Transport building block. Sometimes, ALC combines the LCT building block with a Congestion Control (CC) building block and a FEC building block to achieve a congestion controlled reliable asynchronous delivery.

Asynchronous Layered Coding (ALC, IETF RFC 5775) is a protocol focused on reliable content delivery, designed to provide massive scalability using IP multicast. It's said that is massively scalable because it could manage theoretically about million receivers and because the size of the object to be transmitted could be from kilobytes to hundreds of gigabytes, and the rate of each receiver would be the maximum available bandwidth between the receiver and the sender.

Layered Coding Transport (LCT, IETF RFC 5651) building block provides transport level support for reliable content delivery protocols. It's designed to specially support protocols using multicast and it's compatible with congestion control, that provides multiple rate to different receivers. LCT provides transport level support for massively scalable protocols as ALC, and it supports a variety of main applications as reliable content delivery and streaming applications.

A FLUTE session consists on one or more ALC/LCT channels, where a receiver has to join the channel to start receiving and leaves the channel to stop receiving. Session Description Protocol (SDP) describes the parameters required to begin, join, and leave the FLUTE sessions. Sessions may be started without any knowledge of the FLUTE session content.

Some existing FLUTE implementations:
- Tampere University of Technology → http://www.atm.tut.fi/mad
- University of Bremen → ftp://ftp.informatik.uni-bremen.de/home/logic/flute/
- Institute National de Recherche en Informatique et en Automatique → http://www.inrialpes.fr/planete/people/roca/mcl/mcl.htm
- Nokia (Proprietary) → http://research.nokia.com/publication/10495
2.3.2. Spread Toolkit

“Spread is an open source toolkit that can be used in many distributed applications that require high reliability, high performance, and robust communication among various subsets of members.” It has been designed to enable reliable and scalable applications by encapsulating the challenging aspects of asynchronous networks. The user is linked to a library, a binary daemon which runs on each computer that join the group.

Some services and benefits using this toolkit:

- Reliable and scalable messaging in a group, easy to use, highly scalable depending on the network, supports thousands of groups with different sets of members, emphasis on robustness and high performance, completely distributed algorithms without a central point of failure.

Information of these application can be found in its website: 
http://www.spread.org/index.html

2.3.3. The JGroups Project

It's a toolkit for reliable messaging that can be used to creates clusters. The nodes of these clusters can send messages to each others over LANs or WANs. Main features of these messages are creating and deleting clusters, joining and leaving clusters, membership detection, detection and removal of crashed nodes, sending node-to-node messages...

“The most powerful feature of JGroups is its flexible protocol stack, which allows developers to adapt it to exactly match their application requirements and network characteristics”. In the JGroups website, http://www.jgroups.org/, the specifications and compatibilities are explained.

By mixing and matching protocols, is possible to satisfy many different applications. It's possible to use UDP, TCP, fragment large messages, retransmit lost packages (reliability), failure detection excluding crashed nodes, flow control, membership, encryption and compression.

It is possible to create reliable messaging applications where reliability is a problem. It doesn't have to be implemented by the developer so time is saved and allows using the application in different environments without having to change the code.
2.3.4. OpenPGM

OpenPGM https://code.google.com/p/openpgm/ is an open source implementation of the PGM specification. It's an application for a reliable and scalable multicast protocol, detecting losses, requesting lost transmitted data and notifying an application of unrecoverable losses.

Ubuntu, Debian, OS X, or Windows, as some of available platforms to use this application. Testing OpenPGM has shown on gigabit Ethernet connection speeds of 675Mbps with 1500 byte frames.

2.3.5. UDPcast

To propose a solution for an open source we will run a UDPcast, a simulation software, in order to realize how the platform should work with the new model. UDPcast is a tool of file transfers that can send data to one or more destinations in a LAN. The main advantage of UDPcast is that is very quick to install a lot of similar virtual machines, not having to create them from zero. All the needed information is in its website, https://www.udpcast.linux.lu/.

Main options available in the udp cast are:

udp-sender --async --fec 8x6/64 --max-bitrate 39m --mcast-rdv-addr 224.2.2.1 -mcast-data-addr 224.2.2.2 --interface eth1 -f example.bin

udp-receiver --nosync --mcast-rdv-addr 224.2.2.1 --interface dvb0 -f example.bin
2.4. **Current Tools**

2.4.1. **OpenVPN**

OpenVPN is a highly configurable solution to acquire the necessity of a VPN, it is secure and it can run for free. OpenVPN supports peer-to-peer and multi-client server configurations. It allows many different topologies, exchangeable and adaptable to each network.

Security in OpenVPN is handled by the OpenSSL cryptographic library. Secure Socket Layer (SSL) provides strong security using standard algorithms: AES (Advanced Encryption Standard), Blowfish or TripleDES (3DES). Authentication certificates and encryption are used.

The base to judge the security of an encryption cipher is that if a cipher which stood scrutiny of the security community for many years is considered strong. If this cipher has been failing, they would have found those mistakes. The selected cipher should meet security, performance and availability.

A lot of characteristics of the VPN affects the performance of the system, such as the RTT, the jitter, routers utilization, probability of errors, time between errors, link utilization, the network topology, VPN topology, encryption cypher, compression algorithms... So as far as we have an enormous VPN, a high amount of routers are involved and the VPN number of clients is changing day by day, being difficult to maintain it constantly optimized.

The VPN of this platform consists in a network where are included cinema labs, digital cinemas and the office where the shippings are managed from, by the traffic managers. The expectation of the business is to achieve to connect all the digital cinemas and all the digital cinema labs. We use it to control the servers and their maintenance, and to make possible the transfers, through the data-channel and the back-channel. Tasks to control are keep the servers with the necessary free space to receive properly, to able/unable the different services if it's necessary and to check that the contents are well received. The data-channel is the transmissions from the sender to the receiver: data sending through VPN when it's not receiving properly through the satellite, to handshake at the beginning of a transfer, etc. The back-channel is used to inform from the receiver to the sender: ACKs or NAKs, transmission speed, packets lost, etc. It will be cleared when the different parts of the communication are explained.

This VPN allows us to manage the huge net from an office as if it would be a real private network where all the servers are in the same place.

2.4.2 **DVB-S2**

DVB-S2 (Digital Video Broadcasting – Satellite – 2nd generation) fits enough to the DCP sending platform on satellite. DVB-S2 uses a coding scheme based in modern low-density parity-check (LDPC). This codification is not too many complex it has a special structure, also known has irregular codes. It has modes that allow bandwidth optimization by changing transmission parameters by Variable and Adaptive Coding Modulation (VCM and ACM). A wide modern LDPC is concatenated with a BCH
external code to achieve the reception conditions almost error free in a AWGN (Additive White Gaussian Noise) channel. BCH codes are a class of cycling error-correcting codes that are constructed using finite fields. The acronyms BCH comes from its inventors initials. This external code is used to control the errors in low bit error rate.

For transmission applications, QPSK and 8PSK are proposed. They can be used in non-linear transponders. 8PSK is used on our platform nowadays.

The native sequence for DVB-S2 is based in IP data sent with “Generic Sequence”, including MPEG-4 AVC / H.264 services. Direct entrance of more than one MPEG-2 streams are allowed, and with a compatibility mode, it's possible to use MPEG-TS (Transport Stream).

2.4.3. RBC Protocol (Reliable Bit Cast)

The current sending protocol is the Reliable Bit Cast (RBC). RBC is a protocol developed by the platform enterprise, which is specially thought to transfer large files. Many revisions of the protocol have been done, but the mechanism still having the same ideas about the sender-receivers communication.

Only buffers can emit through satellite channel in order to optimize the data stream; if a buffer is not used, the bit rate cannot be continuous so satellite communication can't be established. The buffer used in this platform, consist in a cache memory that storage received files from the Data Base. Then, these files are send in a smooth continuous stream to the satellite hub. Moreover, the buffer convert this TCP packets received from the controller to UDP and encapsulates the data with MPEG-TS. Then, the satellite hub sends these encapsulated packets to all the receivers via multicast addressing through the satellite.

The MPEG-TS frames are unencapsulated by the DVB driver located in the RxTx server to get the original UDP datagrams, which are injected into the network stack.

Ciphers

Almost all the content is ciphered but it's property of the sender how they do that. It's not an discussable aspect of the protocol. If any content is not ciphered, distributor has the responsibility of it.

Integrity

An MD5 checksum is calculated on every file. The MD5s (MD5 sent) is sent besides the file, so when it's in the RxTx server, it calculates the MD5r (MD5 of the received file). Then, to assure the integrity of the received file, MD5r needs to be exactly equal to MD5s.
This protocol is used to send files through a network using two channels:

1. the **data-channel** is a communication channel going from the sender to the receiver and is used to inform the receiver a new stream has been created, to give the receiver the key to decrypt incoming data in secure mode, to send the receiver checkpoints and to send file data. This channel is either TCP, UDP or UDP over IP.

2. the **back-channel** is a communication channel going from the receiver to the sender. This channel is needed because when sending a file over satellite, the connection is only one-way so the sender cannot retrieve informations sent by the receiver without another channel. Through this channel, the receiver sends to the sender several information such as the data blocks it missed, the bandwidth at which it receives data and the quantity of loss the transport link has. This channel is UDP only.

The RBC protocol defines several packets, each having its own use. Absolutely every packet is prefixed by the RBC header. On the RxTx server, at reception, any packet without a RBC header or with an invalid one is dropped.

The different packets are:

**The Stream Desc** *(data-channel)*: stream desc stands for “stream descriptor”. It is the first packet to be sent at the start of a stream. It is used to describe the stream: it contains the receivers supposed to listen to it, various details on the sent file and information on how to establish the back-channel.

**The Stream Data** *(data-channel)*: stream data are packets containing the information of the file that is being sent.

**The Stream Key** *(data-channel)*: stream key are packets used in secure mode. When this mode is activated, file data is encrypted before being sent. In order to decrypt the data on receiver, the sender gives a crypto-key contained in the stream key packet.

**The Stream Checkpoint** *(data-channel)*: stream checkpoint are packets allowing the use of the checkpoint mode. This mode allows continuous integrity checks during the transfer. Every X bytes (X is adaptable), the sender gives the receiver the MD5 of the data being sent so far in order for the receiver to detect data corruption early. This MD5 is sent in the stream checkpoint packet.

**The Stream Status** *(back-channel)*: stream status packets are sent by the receiver to the sender. They either indicate the file was entirely received well, there was an unrecoverable error during the transfer or data blocks are missing and should be retransmitted.

**The Stream Bandwidth** *(back-channel)*: stream bandwidth packets are sent by the receiver to the sender to inform it about the bit rate the receiver is actually receiving data.
This allows the sender to compute statistics so the agent can know if everything is going well or if there are issues with the transport.

The Stream Switch (data-channel and back-channel): stream switch packets are special packets sent by the sender and the receiver. They are used to switch a receiver of an existing stream to another transport without having to restart the whole stream from the beginning. The packet is first sent by the sender to the receiver and once the receiver is ready it sends back the same packet to the sender to inform it the switch can be done.

2.5. Platform Overview

Analysing this platform, there are some blocks that should be watched:

Front System (Client Interface)

The front system is a group of software that allows the client to perform actions and allows the provider to communicate with their clients. Main functionalities:

- Orders placement
- Tracking and notifications
- Content management

Back System (Internal Interface)

The back system includes tools that allow the responsible team to manage the traffic and the platform. Main functionalities:

- Traffic monitoring
- Access Control Link management
- RxTx park management at business level
- Miscellaneous business logic tasks: update transfers, storage management, streaming management, automatic priority settings, etc...
- RxTx monitoring and support tools

Delivery System

The delivery system is for performing the transfers. Main functionalities:

- Perform uplink (distributor - platform) and downlink (platform - cinemas) transfers
- Transfer route computing: Breaks down a transfer into unique id files.
- Performs transfers

Up to two simultaneous files transfer per client on each route

No limit on the number of global simultaneous file transfers

- Priority handling
If two clients have high priority orders

- Protocol transfer

**Supervision**

The supervision is responsible of synchronizing the state of the RxTx with the central database.

It is basically a repeated script doing the following things:

- Monitors the receivers (at machine level)
- Content management: lists existing contents on the receivers and authorizes the receivers to perform actions (such as deleting content)

It is organized as pools of servers to monitor and cyclically visits them. The latency can be very high.

**Common Database**

The common database is a SQL (Structured Query Language) database serving at the central data bus of the platform.

**Monitoring**

Standard monitoring is provided over the platform. It gathers technical information at system level.
3. Project development:

3.1. Scenario

General view of a sending:

A distributor wants to send a film through this platform, so they need to create the sending. To create it, they need to have loaded the content into the platform cloud database. Once the content is available, they need to select the receiver theatre. That's all the distributor need to do before sending the content.¹

When they launch the sending, the platforms perform the creation of the nanos. Each nano is a unique identifier for one file and one receiver. If the same file is sent to 6 receivers, there are 6 nanos. These nanos will be too useful to identify every transmission.²

These nanos are read by the software who manage the sending, locate the data and assigns each nano to its file. Then, the order is finally created, with the data to send and the receivers of each nano. This software is called “Agent” and is the responsible of sending the data to the buffer.³

Then, this content is sent from the buffer to the sat hub, which is the responsible of sending the data through the satellite to the receivers. To do it, the satellite uses a transponder, which distribute the content on broadcast, but it will be only received properly by the theatres which are compatible with one of the nanos that te content has. If
a dish “not programmed” listen to this content, and it shouldn't, it just drop the content, waiting for content for it.6

To inform the buffer if the content had been received correctly or not, receivers send ACKs or NAKs. That is not sent by satellite, it's sent through the back-channel on VPN. This information is used by the buffer to know if retransmissions of particular content needs to be done7, the buffer informs about that to the Agent8 and the agent write it on the database9.

There is a software that allows the supervision of what's happening on the receivers, to estimate durations of transmission, to make decisions about abortions of transfers, etc. It's possible because the supervision daemon generates the plugins needed to monitiorize the receivers status10, sent from the Agent to each receiver11. Each receiver returns the information required on the plugin querys12. This information of the supervision is not automatized, so always a satellite management team needs to be making decisions about the current transfers to improve the conditions of the next ones.

3.2. Performance of the current platform

To evaluate the platform and be able to compare it with future improved behaviours, it's needed to theoretically work out numerically the transmissions through the network.

To do that, we will base the development of the problem in the annex B, RBC, where the protocol is detailed, and the breakdown of every RBC packet is defined.

The main element to manage the transmission is the buffer. This buffer is the one who receives all the data that need to be sent through the satellite, keeps it in cache memory and release the packets to the satellite hub.

This buffer is in charge of converting the TCP receiving data in UDP packages and encapsulating these packages in MPEG-TS. The buffer is the component that receive all the content from the receivers, getting feedback about their reception conditions. Then, it will be the element where the parameters of the transmission are changed, as for example, modifying the FEC.

The path of content – feedback is closed within the buffer, so it is the element that allow the satellite communication available, because it is on duty of sending a continuous stream to make profit of the satellite, and the one who makes possible that the platform is adaptive, by getting the feedback information and making possible to adapt to the receivers.

To sum up, we need to realize in these steps:

At the beginning of every file we send a Stream Desc packet (Descriptor). After that, one Stream Desc packet every 30 seconds is sent.

After that descriptor, Stream Data is sent. Each file will be fragmented in a proper size in order to be easy to use through the VPN (OpenVPN conditions).

After the whole file, the Stream Key will be sent, to decrypt encrypted information.
3.2.1. Lossless transmission

Looking at the scenario, the main points to take into account are:

The Buffer | VPN | Sat Hub | Receiver

All the files of a DCP will follow a similar course, so we can see how the ASSETMAP is sent, because it will be easier to manage a 5KB file than a 30GB one. So we have a 5KB file.

Before begin with numerical development, we need to realize that OpenVPN will add headers to the original data, so we need to bear in mind how the size of the original need to be if we want to fit the 1500 bytes of the MTU applied by default:

**Unencrypted Packet**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Length</td>
<td>?</td>
</tr>
<tr>
<td>IP Header Length</td>
<td>20</td>
</tr>
<tr>
<td>TCP Header Length</td>
<td>32</td>
</tr>
<tr>
<td>Data Length</td>
<td>X</td>
</tr>
</tbody>
</table>

*figure 2: Unencrypted TCP packet*

Total length depends on the available size in the OpenVPN encrypted packet:

**OpenVPN Encrypted Packet (Default MTU = 1500 bytes)**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Length</td>
<td>1500</td>
</tr>
<tr>
<td>IP Header Length</td>
<td>20</td>
</tr>
<tr>
<td>UDP Header Length</td>
<td>8</td>
</tr>
<tr>
<td>Original IP Header Length</td>
<td>20</td>
</tr>
<tr>
<td>Original TCP Header Length</td>
<td>32</td>
</tr>
<tr>
<td>HMAC Length</td>
<td>20</td>
</tr>
<tr>
<td>IV Length</td>
<td>8</td>
</tr>
<tr>
<td>Sequence number Length</td>
<td>8</td>
</tr>
<tr>
<td>Data Length</td>
<td>X</td>
</tr>
<tr>
<td>Rest (security key)</td>
<td>4</td>
</tr>
</tbody>
</table>

*figure 3: Encrypted UDP packet on the VPN*

Then: Of the 1500 bytes, we have that 1380 bytes destined to data. According to RBC protocol, in each packet, 4 bytes are RBC header, so 1376 bytes of data will be sent in each packet.
First of all, we need to sent the Stream Desc. Following the RBC sizes, it will be:

\[
\begin{align*}
40 \text{ bytes (header)} & \quad 8 \times \text{#Receiver} & \quad 8 \text{ bytes (source info)} \\
\end{align*}
\]

To simulate a real sending, we can assume that this file is going to be sent to 100 cinemas, so the number of receivers is 100.

Stream Desc = \(40 + 8 \times 100 + 8 = 848\) bytes

It will signify that the OpenVPN packet size is:

124 bytes (fields different to data) + 848 bytes (data) = 972 bytes.

Once the Stream Desc is sent, we can begin to send data packets. To do that, we need to fragment the file content.

5kB / 1376 bytes = 4 packets
(5kB → file size, 1376 bytes → data size in a single packet)
  · 3 packets of 1376 bytes; after headers: 3 packets of 1500 bytes
  · 1 packet of 896 bytes; after headers: 1 packet of 1020 bytes

After that, we only need to send the Stream Key to finalize the file transfer:

\[
\begin{align*}
8 \text{ bytes} & \quad 16 \times \text{#Receivers} \\
\end{align*}
\]

8 bytes + 16 bytes x 100 receivers = 1608 bytes (We will need two Stream Key packets)
8 bytes + 16 bytes x 85 receivers = 1568 bytes; with headers: 1492 bytes
8 bytes + 16 bytes x 15 receivers = 248 bytes; with headers: 372 bytes

We can set:
The DSL capacity on 200Mbps.
The delay on VPN 150ms
The Sat Hub processing time 150μs
In the figures, due to the graphics software used, "e" means $10^e$, so $e(-4)$ means $10^{-4}$

To sum up, the total time of a file transfer in this portion of the network (before satellite) will be:

### Stream Desc Time:

$$t = \left[ (\text{fixed fields size} + 8 \times \#\text{ofReceivers}) \times 8 \right] / (200\text{Mbps} \times 1024^2 \times \text{bits to Megabits})$$

$$t = \left[ (48 + 8 \times \#\text{ofReceivers}) \times 8 \right] / (200 \times 1024^2)$$

*Every 30 seconds, a Stream Desc packet will be sent*

### Stream Data Time:

File Size / 1376 bytes = #completePackets

$$t = \#\text{completePackets} \times 5.72 \times 10^{-5} + [(\text{FileSize}-1376 \times \#\text{completePackets}) \times 8] / (200 \times 1024^2)$$

### Stream Key Time:

#receivers / 85 = #completeStreamKeyPackets

$$t = \#\text{completeStreamKeyPackets} \times 5.69 \times 10^{-5} + (8 + 16 \times \#\text{Receivers}) \times 8 / (200 \times 1024^2)$$
Once it's on the Sat Hub, the DVB-S2 encapsulation to send through the satellite is needed. To do this, we need to see how the data is managed.

The size of the DVB-S2 packet is:

10 bytes DVB-S2 Header | Data | 4 bytes (CRC)

FEC used is 5/6, so 1/6 of the Data size will be redundant.

If the OpenVPN packet size is 1500 bytes, the DVB-S2 packet will be:

\[
(10 + 1500 + 4) \times \frac{6}{5} = 1817 \text{ bytes}
\]

To convert from bytes to Mbps, at 50Mbps:

\[
\text{time} = \frac{(10 + \text{Size of OpenVPN Packet} + 4) \times \frac{6}{5}}{\text{TOTAL SATBYTES}} = \frac{\text{SATBYTES} \times 1 \text{ MB} \times 8 \text{Mb} \times 1 \text{s}}{1024^2 \text{B} \times 1 \text{MB} \times 50 \text{Mbps}} = \text{[s]}
\]

In this example:

Stream Desc = 972 bytes → 1184 bytes (after encapsulation) → 1.8x10^{-4}s

![Figure 5: Lossless satellite transmission](image)
3.2.2. Analysis of the losses

In the transmission between buffer and sat hub, we can assume that it cannot exist losses, because all the transmissions are cable connections, being very strange to have a bit error rate different from zero.

On the satellite transmission, by default, the FEC makes the transmission almost without losses if it's set with accuracy and the conditions are proper. We are using 5/6 because with this FEC we can assume it's error free in good reception conditions. If we wanted, we could set a more redundant FEC but if the receiving conditions are proper, it's not needed.

So we assume that we don't have losses in the satellite, but the weather factors are very important and they introduce many losses.

If clouds are being formed between the satellite and a dish, it will mean that this receiver will receive worse than it should and it's going to lose packets. It's not implemented the adaptive sending possibility, so if some receiver is having problem with its bandwidth, it won't be able to receive. After the whole file has been sent through the satellite, each receiver will request for the missing packages in order to fill the blanks in the current file. Until all the receivers get the file, a new file through the satellite cannot be sent.

Summarizing, the FEC will avoid losses through the satellite always the reception conditions are proper, but if receivers are in reception troubles, all the needed packets will be retransmitted at the end of the first transmission (before the requested files are sent).
3.3. **Current Platform Problematic**

One of the first comments that have to be made is that the base code of the platform can be very hard to maintain. It would maybe be better to use a commonly worldwide known protocol. Maybe it wouldn’t be as accurate to our needs as RBC is, but it’s obvious that many bugs should disappear. Furthermore, it would be easier to expand the capabilities of the platform (It would be really hard to adapt the RBC protocol for a new purpose but it’s sure that some existent protocol works almost perfectly). It will be nice to study the possibilities in this area.

Transfers cannot be finished without the intervention of the supervision because if a server is not receiving at all (0Mbps, for example if it's offline) all the platform would be collapsed. The necessity of human supervision has no sense taking into account the amount of daily transfers. Too many people must be monitoring the platform everyday in order to control the progresses. In fact, dependencies are present in many procedures and we should try to improve it. For example, using the DVB-S2 characteristics of Variable and Adaptive Coding and modulation.

When we are dealing with real conditions, we can’t achieve the theoretical sending time of a DCP. That’s why we are connected to many receiving servers and the conditions are changing all the time. The main issues are: the bad weather is disturbing to the reception, the DSL is not working fine and that the receiver has lost the DSL connection (server turned off, VPN not properly started...).

These issues are very common due to many servers are connected, and when a server is not receiving well, something needs to be done, because due to the protocol, the transfer is not finalizing if one server or more hasn't received the whole file. So if a server has lost the DSL connection during the night and it has not been recovered, the transfer is probably blocking the next file and it’s monopolizing the platform, not being used.

As far as we need a DSL connected to the server located in the cinema in order to connect to the VPN, we have the possibility of sending data through the VPN. The swap needs to be done manually, and it’s hard to manage if many servers are failing. It's a hard work, because it has to be known the quality of the DSL service; if in a cinema they have optical fiber it's easy to make decisions, but if they only dispose of about 1 or 2 Mbps, swapping to VPN is a difficult choice. We wait for the recovery on satellite blocking the platform for a period or we swap it to VPN and it won't probably finish for the day of the première.

There exist the last chance to get that the DCP arrives to the cinema, and it's by a courier. It's outdated but when it's necessary, the delivery enterprises work well and they use to send it in less than 24h and with a very low cost.
3.4. Improvements Proposal

In order to make the new proposals for the current platform, we will divide them in protocol improvements and topology improvements.

3.4.1. Protocol Improvements Proposal

Taking into account that some parameters of the platform are unused, the main purpose is to use an adaptive protocol in order to improve the satellite efficiency.

As far as the receiver gives feedback about its reception conditions, the buffer only needs to manage this information to fit to receiver conditions and get better results of efficiency aside the time of sending, exploiting the DVB-S2 parameters to be adaptive, because at the end, sending with retransmissions will be longer than if we adapt to the problems. It's not possible to adapt to all receivers, and that's because if one receiver is not receiving (0Mbps), buffer is not going to stop just because one receiver is not working. And the same happens if the worse receiver is running at 1Mbps, 5Mbps... so we need to set the decisions on the sat hub to manage it.

– How can the source adapt to the receivers?

First of all, we need to clarify why the packets are lost. The reason is that the bad weather and other possible channel issues affect enough to the receivers so they don't receive properly, so the FEC that the source is applying to the data stream is not enough for these receivers that have reception problems. These receivers are losing too many bits and the FEC is not redundant enough to recover these bits so the integrity of these packets is affected. If this FEC is not enough, this receiver sends NAKs to inform the source that these packets need to be retransmitted.

When the receivers send feedback to the source, they send information about how many packets they have lost. This information is not used in this platform for the moment, but it should be easy to implement the interpretation of this information due to the adaptive property of the DVB-S2, and modify the FEC to achieve a better success rate of packets transfer.

So if we want to improve the performance of the platform, we can use the feedback information to adapt the FEC to the receivers. Seems obvious that increasing the FEC will be better than if we have to retransmit packets, but it has to be studied to ensure that the behaviour of the platform is going to be improved.
3.4.2 Topology Improvements Proposal

A possible way to face this real-conditions problem is to revise the topology of the whole platform. Nowadays we have a Master (sending server) and all the destinations are Slaves. It would be possible to change the rules of the game. We could have a Master and Slaves, but we could have SubMasters too.

![Figure 7: Topology modified scenario with regions](image)

The delivery begins as always, with all the needed receivers. The idea is that the Master sends each packet to all the destination servers, with the necessary protocols to try that at once all the servers get the information. These receivers will be packed in regions.

These regions won’t be used if all the receivers can receive properly, and if everything goes as well as it should, the transfer won’t have any change in comparison to when only existed Master and Slaves.

![Figure 8a: Explanation of regions distribution](image)
But when a receiver begins to fail a lot, it becomes a slave. So one of the receivers of its region (one receiving properly) becomes the SubMaster. After that, the transmission won't change, all the receivers server will receive content as before, but the slaves won't request for retransmissions any more. The important achievement is that the SubMaster of each region gets all the packets. These retransmission will be done between the receivers of the region that have the content. The most receivers have the complete file, the best, because it won't be needed so much retransmissions.

The main idea of this method is that the Master is free as soon as possible in order to be able to focus in other sendings. Of course, the SubMaster-Slaves are monitored too, in order to be sure that everybody get all their packets.

To manage the regions retransmission, it would be intelligent to use a p2p peer-to-peer protocol. Protocols based in p2p consist in a computer network where are located some nodes where each one behaves as equal to the others, simultaneously as clients and servers to the whole network. A priori, it wouldn't be necessary to set a SubMaster. The SubMaster won't be the only one who retransmit the information, but it will be set with this category to communicate with the Master, in order to ensure that all the packets are in the p2p network. A good p2p option would be BitTorrent. BitTorrent is a protocol specially designed to the file exchange peer-to-peer through DSL. It's one of the more common protocols for the high size file transfers.

Seems to make sense to set the regions geographically, but in the practice, so many times the transfer fails due to the weather. So set each region with receivers of all the territory will be an assurance of almost one server of each region will receive properly.

We could settle the SubMasters strategically in servers that we know that are more robust. In fact, a mechanism of SubMaster election can be done by studying the receiving servers in the moment of the transmission beginning.
Taking into account that usually about only 15% of receivers (or less) have problems on receiving properly, we can set regions of about 10 receivers. It will ensure that meanly, each region will deal with one or two failing slaves.

It will maybe increase the Total DCP Receiving Time, but it will decrease the Total DCP Sending Time from the source point of view.

To sum up, the main advantage of this improvement proposal is decrease the total retransmission time from the source, in order to release its occupation so it can begin to send another file.
4. Results

4.1. Results by improving the protocol

As proposed before, we need to realize how the FEC can be modified to be more efficient. To do that, we establish the case of a receiver losing packets. Always a receiver has a higher packet loss rate than the redundancy, it will mean that retransmission will be needed or we need to increase the FEC.

If we have a 5/6 bits, 1 of every 6 bits is redundant, so if the receiver gets 5 of every 6 bits, it will be able to repair the packet by itself. It means that if a receiver is losing 17% of bits or more, it's sure that it will lose packets. If we have an initial FEC of 5/6, we can increase the FEC by adding FEC in the buffer, so the encapsulated packets from the buffer have an implicit FEC in order to get a better result after the satellite transfer.

So, if the source increase the FEC, the transmission time of the whole file will be increased too. If we initially use a 5/6 FEC, of every 100 bits, 83 bits are of original data and the other 17 are redundant, but if we use a 3/4 FEC, of every 100 bits we send 75 bits are original data and 25 are redundant.

To clarify this point, we cannot modify the FEC on the sat hub, because the operator won't give us the freedom of modifying the FEC continuously. So the 5/6 FEC will be the basic FEC of the transmission, but all the necessary added FEC will be set at the buffer. It means that when we say that we are increasing the FEC from 5/6 to 4/5, the buffer is setting at a lower level a FEC in order that combining this added FEC with the 5/6 base FEC will result a 4/5 FEC.

If we have the 1500MTU packets, when they run through satellite, we have:

\[
(10 \text{ bytes } \text{*header*} + 1500 \text{ bytes } \text{*data*} + 4 \text{ bytes } \text{*CRC*}) \times (1 / \text{FEC})
\]

So if we compare 5/6 FEC with 4/5:

- using FEC = 5/6:
  
  \[
  1817 \text{ bytes} \times 100 \text{ packets} \times (6/5) = 218040 \text{ bytes}
  \]
  
  at 50 Mbps → 33.27 ms

- using FEC = 4/5:
  
  \[
  1817 \text{ bytes} \times 100 \text{ packets} \times (5/4) = 227125 \text{ bytes}
  \]
  
  at 50 Mbps → 34.57 ms

*These calculations are made as if the FEC were added at the sat hub. It's not the way it works, but at the end, with high file sizes, the results would be nearly the same, because the amount of bits to send are barely the same.
It means that after 100 packets, the difference between 5/6 FEC packet and 4/5 FEC one, will be 1.3 ms.

If we calculate the difference between the retransmission time and the FEC adding time,

\[
1.3\text{ms} / 0.33\text{ms} = 4 \text{ packets}
\]

So if the receiver is losing more than 4% of the packets, it will be better to increase the FEC to 4/5.

If we compare 5/6 FEC with 4/3:

- **using FEC = 5/6**:
  
  1817 bytes × 100 packets × (6/5) = 218040 bytes
  
  at 50 Mbps → 33.27 ms

- **using FEC = 3/4**
  
  1817 bytes × 100 packets × (4/3) = 242267 bytes
  
  at 50 Mbps → 36.97 ms

Taking into account that each packet needs 0.33ms to be retransmitted, if 12 or more packets of these 100 are lost and have to be retransmitted, it's better to increase the FEC. So, if we have a 12% of packets loss, increase the FEC from 5/6 to 3/4 is a good idea.

If we calculate the transmission time of a video file, about 30 GB and we compare the previous results, we can realize when it's necessary to increase the FEC:

30 GB ~ 3.22x10¹⁰ bytes

So we need: 3.22x10¹⁰ bytes / 1817 bytes = 17728264 packets

Transmission time with 5/6 FEC:

Packet time:

1817 × (6/5) = 2181 bytes

2181 bytes × 8 bits / (50 Mbps × 1024² bits) = 3.3279x10⁻⁴ s

Total time = 17728264 × 3.3279x10⁻⁴ s = 5900 s → 1 h 38 m 20 s
Transmission time with 4/5 FEC:

\[ 1817 \times \left(\frac{5}{4}\right) = 2272 \text{ bytes} \]

\[ 2272 \text{ bytes} \times \frac{8 \text{ bits}}{(50 \text{ Mbps} \times 1024^2 \text{ bits})} = 3.4668 \times 10^{-4} \text{ s} \]

Total time = \(17728264 \times 3.4668 \times 10^{-4} \text{ s} = 6146 \text{ s} \rightarrow 1 \text{ h } 42 \text{ m } 26 \text{ s} \)

Transmission time with 3/4 FEC:

\[ 1817 \times \left(\frac{4}{3}\right) = 2423 \text{ bytes} \]

\[ 2423 \text{ bytes} \times \frac{8 \text{ bits}}{(50 \text{ Mbps} \times 1024^2 \text{ bits})} = 3.6972 \times 10^{-4} \text{ s} \]

Total time = \(17728264 \times 3.6972 \times 10^{-4} \text{ s} = 6555 \text{ s} \rightarrow 1 \text{ h } 49 \text{ m } 15 \text{ s} \)

So, as studied before, if we are over 4% of packet loss, it will be better to increase the FEC to 4/5. A 4% of retransmission would mean 236s, about 4 minutes, and the transfer times would be the same.

The same would happen with 3/4 FEC if the packet loss is over 12% with a 5/6 FEC.
4.2. Results improving the topology

The adaptive protocol proposal work fine if the % of packet loss isn't very high. If we have to deal with 1/2 FEC, or more than the half of packets needs to be retransmitted, it's better to set the bad receiving servers as Slaves and the source should not take care about its retransmission requests.

To set a parameter limit, if a receiver is dealing with 25% or more packet loss, it should be set as a Slave. If the 3/4 FEC do not satisfy the receiver necessities, then, the bitTorrent protocol will be the one in charge to make this receiver get all the needed information.

If the source takes 1h 49m 15s to send a 30GB file instead 1h 38m 20s of because of the modification of the FEC, we can set that a whole film, that could be about 170GB, would mean about 10h 19m 5s instead of 9h 17m 13s.

We can assume an hour of extra delay in a whole film, but to assume more than that, it would be too much. That's why the FEC won't be more redundant than 3/4.

But if just one server would need a higher FEC, it wouldn't be optimum to set it, because if all the other servers are receiving well, this extra hour won't be assumed just for a receiver. So, if less than 5% request for a higher FEC, it won't be granted, because it doesn't worth the extra time effort to satisfy less than the 5% if they can get the content from its p2p region receivers.

To sum up, if more than 5% of the servers request to increase the FEC, it will be increased, but never higher than 3/4. If some receiver tries to request more than a 3/4 FEC or less than 5% of servers need this increase, they will be set as Slaves and they will wait to the bitTorrent retransmission.
5. **Budget**

To make the budget of this project two main parts need to be taken into account:
- The design of the new model of deliveries
- The implementation of carrying out this new model of deliveries

The design of the new model, if it's done by workers of a digital cinema content delivery enterprise, if the parameters are studied, optimized and tested, should be about 200 hours.

The real implementation of the project, typing the code, testing and optimizing the software and training the receivers to manage the new model, should be about six months, that means 720 hours.

The final cost of the accomplishment of the project will depend on the salary of the workers, but if we set this salary, for example, in 20€ per hour, this project would cost about 18.400€.

Moreover, this project is thought not just in a technical way, but in the easiest and cheapest one, because the implementation of solutions of the content delivery problems mean a deduction of the total costs. The most satellite deliveries are successful, the least hard drives are needed. The cost of the hard drives depend on its price and its lifetime, and the satellite has a fixed cost, so everything that means improvements on the non-physics deliveries are benefits in time, faithfulness and final costs.

Furthermore, using a widely known and used protocol signify a reduction of hours of work in the implementation of the new purposes of the protocol in the future. These non-used hours on the protocol design will be able to be used to work in another areas, as users interfaces, investigation of new possible solutions for current reported issues from the distributors or developing new projects in the cinema area networks.

All the software used is open source code, so it doesn't mean any cost. After that improvements, a person isn't needed full time to manage this transmissions, so it will signify that more than the half of an engineer salary is saved every month after the development of this project.

In addition, everything that mean a high rate of success in satellite deliveries, will have repercussions in the judgement of all the clients, being efficient and trustworthy, so the loyalty of the current clients will be achieved and new clients will be obtained if the results show that the performance has a high rate of success.
6. **Environment Impact**

The environmental impact of these project seems to be not very important, but the digital cinema has been introduced many benefits.

In terms of production, the digital cinema labs has finished with the traditional cinema labs of 35mm. These 35mm labs meant a lot of chemical products that are harmful to the environment, because eliminate the wastes of these chemical products is a hard and a long term job. For example, is thought that the darkroom workers were more susceptible to develop some form of cancer. Some photo chemicals can lead to acute problems such as burns, dermatitis, dizziness, vomiting, asphyxiation and central nervous system failure. Now, with the digital "revealing" techniques, of course that all of that is obsolete and no hazardous impact is possible, because chemicals are not used any more.

Another way decreasing the environment impact is on the delivery of the content. When the 35mm reels were sent to cinemas, they needed to be sent by common transport. Distributing cinema content by road (and by plane or ship too) generates a high amount of CO$_2$ emissions. Once this process has been modified to the electronic delivery, these emissions don't exist any more.

The impact related to the digital cinema, would be reduced to the energetic impact of the servers and projectors. These equipments don't waste much more energy than a computer, so the impact of every one of them is not worrying. Of course it's significant and everything adds when the environment is related, but comparing it with the 35mm processes and the CO$_2$ emissions of reels deliveries, I think it worth.
7. **Conclusions and future development:**

As a general idea, I think that making this platform adaptive will save a lot of money in the future, in terms of not being necessary a satellite controller to swap connections between satellite and VPN, and in terms of being efficient so maybe if it's well optimized a lower bandwidth is needed, so the transponder of the satellite can be shared and the fix cost is very lower. Further, the automation of the platform would stand for a better rate of deliveries success, so it would be highly profitable to make new clients.

About future development, the idea is to implement the automation processes on the platform, setting parameters to decide making decisions properly. The automation process will have to take into account different points that in the practice maybe differs from the theoretical computation, as choosing the best SubMaster of each region, to adapt the FEC depending on if there are more films to send before the current one, or depending on the affected number of receivers taking into account the première date, and notifying automatically to the transfer control manager and to the theatre operator if the connection has been lost, in order to speed up the receiver recovering.

This automation of the platform would signify a solid improve of the performance of the platform, because being adaptive will optimize the behaviour results of the transfers; and apart of that, it would mean that the platform control manager could expend his time in improving the behaviour of the platform and not having to look after every transmission.

To sum up, the whole idea of the project and the future development, is to optimize and automate, as much as possible, the whole platform and all the related processes, following the trend of being every day more and more computer dependant, due to their efficiency and its performance results. Of course, every time a person job is converted in a computer job, others jobs are created to the people, for example, on creating the needed software for this automated work or on developing new operations systems possible because of the automation.
Bibliography:

[9] NACK-Oriented Reliable Multicast (NORM) Transport Protocol, IETF RFC 5740, November 2009.2039455
[12] Forward Error Correction (FEC) Building Block, IETF RFC 3452, August 2007
[22] MAD Project's (FLUTE), Tampere University of Technology: http://mad.cs.tut.fi/
[26] UDPcast webpage: https://www.udpcast.linux.lu/
[27] Asynchronous Layer Coding (ALC) Protocol Instantiation, IETF RFC 5775, April 2010
[28] Layered Coding Transport (LCT) Building Block, IETF RFC 5651, October 2009
Appendix A

Complete DCP example:

CPL_YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV.xml
CPL_YvesSaintLaurent_FTR_S_FR-ES_INT_51_2K_VER_20140714_YBA_IOP_OV.xml
  – This means that two films are included in this DCP. With this DCP, Spanish Version and VOSE can be played.

ASSETMAP
PKL_YvesSaintLaurent_fcef048d-4942-4526-bba1-416d84615adc.xml
VOLINDEX

p2k_4f250c0c-19b8-4cf4-bd42-8050a57386b0_video.mxf
p2k_5b68e180-5ca6-4e2c-86f6-086edca457ac_video.mxf
jp2k_5e3fd4b-6668-4722-983b-65f3b5bd3bb_video.mxf
jp2k_8f5b7c9f-145b-499e-be80-43337e7b7b66_video.mxf
jp2k_b23cd4f7-498c-452a-af73-1351a59ae88a_video.mxf
jp2k_d8f536e1-4f78-42c6-8b6d-caa03f558d454_video.mxf
-Video segmented in reels

wav_00641a7f-05ec-48fd-b96e-9176f0588c41_audio.mxf
wav_61bdc8b7-82ae-47b9-8d97-eed363d198a_audio.mxf
wav_996473ff-34ef-459c-81f1-6af7b161a8fc_audio.mxf
wav_b5f1f0bc-dfff-4bfe-a1c6-eb2bfaa50c93_audio.mxf
wav_d65bf9f4-4ea2-4a5-22be-79eaf689519_audio.mxf
wav_f1414f50-0ca0-4b98-86e3-62dfc0ee6656_audio.mxf
-Audio of the original version

YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_08.mxf
YvesSaintLaurent_FTR_S_FR-ES_INT_51_2K_VER_20140710_YBA_IOP_OV_01.mxf
YvesSaintLaurent_FTR_S_FR-ES_INT_51_2K_VER_20140710_YBA_IOP_OV_02.mxf
-Video of the Spanish version that differs from the original version

YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_audio_03.mxf
YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_audio_04.mxf
YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_audio_05.mxf
YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_audio_06.mxf
YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_audio_07.mxf
YvesSaintLaurent_FTR_S_ES-XX_INT_51_2K_VER_20140725_YBA_IOP_OV_audio_08.mxf
YvesSaintLaurent_FTR_S_FR-ES_INT_51_2K_VER_20140710_YBA_IOP_OV_audio_01.mxf
YvesSaintLaurent_FTR_S_FR-ES_INT_51_2K_VER_20140710_YBA_IOP_OV_audio_02.mxf

- Audio of the Spanish version
Appendix B

RBC Protocol

RBC2 Header
The RBC header is a header present at the beginning of every RBC packet. Absolutely every packet is prefixed by the RBC header, any packet without a RBC header or with an invalid one being dropped.

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>version</td>
</tr>
<tr>
<td>8</td>
<td>type</td>
</tr>
<tr>
<td>16</td>
<td>length</td>
</tr>
<tr>
<td>24</td>
<td>length</td>
</tr>
</tbody>
</table>

As shown by the figure 9, an RBC header only contains three fields:
1) version (8 bits): contains the version of the RBC protocol being in use. This is used for compatibility purposes. As of January 7, 2013, this field is always equal to 0.
2) type (8 bits): contains the type of the packet (stream desc, stream data, stream key, stream checkpoint, stream status, stream bandwidth or stream switch).
3) length (16 bits): contains the length of the RBC packet.

The following table shows the different values the type field of the RBC header accepts as of January 7, 2013:

<table>
<thead>
<tr>
<th>stream desc</th>
<th>0x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream data</td>
<td>0x1</td>
</tr>
<tr>
<td>stream status</td>
<td>0x2</td>
</tr>
<tr>
<td>stream key</td>
<td>0x3</td>
</tr>
<tr>
<td>stream checkpoint</td>
<td>0x4</td>
</tr>
<tr>
<td>stream bandwidth</td>
<td>0x5</td>
</tr>
<tr>
<td>stream switch</td>
<td>0x6</td>
</tr>
</tbody>
</table>
Stream Desc
Stream Desc packets ("stream descriptor") are packets used to describe the stream. The first packet sent when a new stream is created is a stream desc. Receivers only listen to a stream on their side when they have previously received a stream desc telling them they should listen to it. Then, as the receivers of a stream or the parameters of a stream can change over time, stream desc packets are sent at regular interval to update these changes (usually 30 seconds). The figure 11 describes the contents of a stream desc packet.

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0 – 15</th>
<th>16 – 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>stream id</td>
</tr>
<tr>
<td>32</td>
<td>flags</td>
<td>size…</td>
</tr>
<tr>
<td>64</td>
<td>…size…</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>…size</td>
<td>MD5</td>
</tr>
<tr>
<td>128</td>
<td>…MD5…</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>…MD5</td>
<td>FEC Algorithm</td>
</tr>
<tr>
<td>256</td>
<td>nack port</td>
<td>unused</td>
</tr>
<tr>
<td>288</td>
<td>unused</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>block size</td>
<td>dest list size</td>
</tr>
<tr>
<td>352</td>
<td>dest1</td>
<td></td>
</tr>
<tr>
<td>384</td>
<td></td>
<td></td>
</tr>
<tr>
<td>416</td>
<td>dest2</td>
<td></td>
</tr>
<tr>
<td>448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>480</td>
<td>…</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>544</td>
<td>name size</td>
<td>filename</td>
</tr>
<tr>
<td>576</td>
<td>source size</td>
<td>source</td>
</tr>
</tbody>
</table>

Figure 11: Stream Desc breakdown

Stream id (32 bits): this field contains an ID referencing the stream. It is a 32 bits integer randomly generated by the sender. It allows to keep track of the streams.

Flags (16 bits): this field is a bitmask allowing to set some options on the stream. The following flags are supported as of January 7, 2013:
- desc size (0001): the size of the file is available in the stream desc packet
- desc md5 (0010): the MD5 of the file is available in the stream desc packet
- desc secure (0100): the stream is in secure mode, meaning that the stream data packets are crypted
- desc checkpoints (1000): checkpoint mode is active, meaning that the receivers will get stream checkpoints packets during the transfer

Size (64 bits): this field contains the size of the file. Although this field is always present in the stream desc packets, the actual size of the file is only set when the sender sent the
whole file at least once during the stream, meaning this field is set when the sender only has to handle retransmits, otherwise this field is set to 0.

**MD5 (128 bits):** this field contains the MD5 checksum of the file. Although this field is always present in the stream desc packets, the actual MD5 of the file is only set when the sender sent the whole file at least once during the stream, meaning this field is set when the sender only has to handle retransmits, otherwise this field is set to 0.

**FEC algorithm (16 bits):** this field is unused. It is supposed to be used when RBC will support Forward Error Correction.

**nack port (16 bits):** the UDP port the sender is listening to for the back-channel. Receivers are supposed to send stream status, stream bandwidth and stream switch packets to this port.

**unused (48 bits):** reserved bits for future usage. Currently unused.

**block size (16 bits):** the size of the blocks sent. A block is a chunk of the file sent in a single stream data packet. This value is set at the start of the stream and can’t be changed, even after a switch.

**dest list size (16 bits):** the size of the recipients list for the stream. As the number of recipients of a stream can vary, this field is needed to tell the receivers the size of the next field in order for them to parse it. “Dest” is a short for the French word “destinataire” meaning “recipient”.

**dest list** (variable, equals to the value of dest list size): each sender and each receiver has an RBC ID. This ID is a variable-length string and is used to identify each machine using RBC. The dest list is a list of recipients containing these IDs. As RBC IDs are variable in length, each entry in the dest list is actually the first 64 bits of the MD5 hash of an RBC ID, rendering them fixed in length and easy to parse.

**name size (16 bits):** this field contains the size of the filename the receivers must use to save the file data

**filename** (variable, equals to the value of name size): this field contains the name of the file the receivers must use to save data.

**source size (16 bits):** this field contains the size of the sender’s RBC ID.

**source** (variable, equals to the value of source size): this field contains the sender’s RBC ID. Contrary to the RBC IDs in the dest list, this one is complete and in plain text, not a part of an MD5 hash.

### Stream Data
Stream Data packets are the most used packets during a stream as they are the packets containing the file being sent. The size of a stream data packet is always the same for a given stream, it can’t change during the stream’s life. As shown by the figure 12, stream data packets contains only three fields:

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0 – 15</th>
<th>16 – 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stream id</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>block number</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>data</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*figure 12: Stream Data breakdown*
stream id (32 bits): this field contains an ID referencing the stream. It is a 32 bits integer randomly generated by the sender. It allows to keep track of the streams.

block number (64 bits): this field contains the number of the data block contained in the packet. A block is a chunk of the file sent in a single stream data packet. This fields allows the receiver to know the offset in the file of the following data.

data (variable, equals to the value in the block size field of the stream desc packet): this field contains the actual file data to be written on disk. If using secure mode, this data is encrypted.

Stream Key
Stream Key packets are only used in secure mode. Secure mode is a mode where file data and only file data in a stream is encrypted. The RBC2 encryption uses a two-steps method: a symmetric encryption algorithm, RC4 as of January 7, 2013, is used to crypt the data. The RC4 key is sent in a stream key packet beforehand. The RC4 key contained in the stream key is crypted using RSA, an asymmetric encryption algorithm. As shown in figure 13, a stream key packet contains six fields:

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0 – 15</th>
<th>16 – 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stream id</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>algorithm</td>
<td>key length</td>
</tr>
<tr>
<td>64 - 96</td>
<td>dest</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>key size</td>
<td></td>
</tr>
<tr>
<td>160 - 192</td>
<td>key</td>
<td></td>
</tr>
</tbody>
</table>

figure 13: Stream Key breakdown

stream id (32 bits): this field contains an ID referencing the stream. It is a 32 bits integer randomly generated by the sender. It allows to keep track of the streams.

algorithm (16 bits): this field contains the ID of the used algorithm for crypting the file data. As of January 7, 2013, the supported algorithms are:
- RC4 (0x0)
- Salsa20 (0x2)

key length (16 bits): length of the RSA encrypted key, in bytes

The remaining fields of the stream key packets are repeated for each recipient of the stream. This repeated fields are:

dest (64 bits): this field contains the first 8 bytes of the receiver’s RBC ID’s MD5 hash.

key size (32 bits): this field contains the size of the key used to crypt the file data.

key (variable, equals to the value in the key size field): key used to crypt the file data.

Stream Checkpoint
Stream Checkpoint packets are only sent in checkpoint mode. The checkpoint mode is used to detect data corruption early so that the receiver with corrupted data can ask for the retransmit of the corrupted blocks. Every X sent blocks, X being configured on the sender side, the sender gives the receivers the MD5 of the file from the beginning to the checkpoint block. These MD5 are sent in stream checkpoint packets. As shown in figure 14, stream checkpoints are composed of two fixed fields and then a list of checkpoints.
Checkpoints are composed of a block number and the MD5 checksum of the file from the beginning of the file to this block number.

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0 – 15</th>
<th>16 – 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stream id</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>checkpoint list size</td>
<td>block number…</td>
</tr>
<tr>
<td>64</td>
<td>…block number…</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>…block number</td>
<td>MD5…</td>
</tr>
<tr>
<td>128</td>
<td>…MD5…</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>…MD5</td>
<td>block number…</td>
</tr>
<tr>
<td>256</td>
<td>…block number…</td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>…block number</td>
<td>MD5…</td>
</tr>
<tr>
<td>320</td>
<td>…MD5…</td>
<td></td>
</tr>
<tr>
<td>352</td>
<td></td>
<td></td>
</tr>
<tr>
<td>384</td>
<td></td>
<td></td>
</tr>
<tr>
<td>416</td>
<td>…MD5</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 14: Stream Checkpoint breakdown*

**stream id (32 bits):** this field contains an ID referencing the stream. It is a 32 bits integer randomly generated by the sender. It allows to keep track of the streams.

**checkpoint list size (16 bits):** this field contains the size of the list of checkpoints.

**checkpoints (64 bits + 128 bits repeated X times):** a list of pairs of checkpoints, a checkpoint being composed of:

- **block number (64 bits):** this field contains the offset of the block the MD5 computation went until.
- **MD5 (128 bits):** this field contains the MD5 checksum of the file from the beginning of the file to the block indicated by the previous field block number.

**Stream Status**

Stream Status packets are used by receivers to report their status to the sender. There are 3 types of stream status:
success: the file was entirely received by the receiver and contains no error
error: there was an error during the file copy (no space left on device, MD5 mismatch, ...)
nack: some packets were lost or dropped during the transfer and necessitate a retransmit

Nack packets are used to ask for retransmit. They are composed of pairs representing ranges of packets that were either lost or dropped by the receiver. Stream status packets are sent when the stream is finished on the receiver’s side (“success” status), when an error occurred on the receiver’s side (“error” status), when a stream desc is received (“Nack” status) and every 30 seconds so that if a receiver gets no packet from the sender during a long period of time due to poor satellite conditions, the receiver still pings back the sender to tell the sender it is still alive (“nack” status). The fields of stream status packets are:

**stream id (32 bits):** this field contains an ID referencing the stream. It is a 32 bits integer randomly generated by the sender. It allows to keep track of the streams.

**status (8 bits):** this field contains the type of the stream status packet, as explained earlier.

**nack list size (16 bits):** this field contains the size of the list of nacks (0 if status is “error” or “success”, can also be 0 if no data packet was lost or dropped)

**nack list (64 bits + 64 bits repeated X times):** this field contains a list of pairs of nacks (empty if status is not “nack”), a nack being composed of:

- **range start (64 bits):** this field contains the block number of the first missing packet of the range
- **range end (64 bits):** this field contains the block number of the last missing packet of the range

**error message size (16 bits):** this field contains the size of the error message (0 if status is not “error”).

**error message** (variable, equals to the value of error message size field): this field contains the error message of the receiver
source size (16 bits): this field contains the size of the RBC ID of the receiver sending the stream status packet.

source (variable, equals to the value in the source size field): this field contains the RBC ID of the receiver sending the stream status packet.

last descriptor timestamp (32 bits): this optional field contains the timestamp of the last stream desc packet the receiver got from the sender.

Stream Bandwidth
Stream Bandwidth packets are sent by receivers to give the sender informations about their reception. It allows the sender to compute various statistics about the quality of the transport used to reach each receiver. As shown by the figure 16, the fields of a stream bandwidth packet are:

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0 – 15</th>
<th>16 – 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stream id</td>
<td></td>
</tr>
<tr>
<td>32, 64</td>
<td>bandwidth</td>
<td></td>
</tr>
<tr>
<td>96, 128</td>
<td>received blocks</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>source size</td>
<td>source</td>
</tr>
<tr>
<td>192, 224</td>
<td>written blocks</td>
<td></td>
</tr>
<tr>
<td>256, 288</td>
<td>write bandwidth</td>
<td></td>
</tr>
</tbody>
</table>

Figure 16: Stream Bandwidth breakdown

stream id (32 bits): this field contains an ID referencing the stream. It is a 32 bits integer randomly generated by the sender. It allows to keep track of the streams.

bandwidth (64 bits): this field contains the general reception bandwidth of the stream for this receiver.

received blocks (64 bits): this field contains the number of block received by this receiver. This number takes into account blocks received multiple times due to retransmit.

source size (16 bits): this field contains the size of the RBC ID of the receiver sending the stream bandwidth packet.

source (variable, equals to the value in the source size field): this field contains the RBC ID of the receiver sending the stream bandwidth packet.

written blocks (64 bits): this field contains the number of blocks received at least once. This is a number of blocks written to disk so far.

write bandwidth (64 bits): this field contains the effective bandwidth for this receiver, effective meaning data blocks the receiver hasn’t received so far due to retransmit.

Stream Switch
Stream Switch packets are used both by the sender and by the receiver. These packets allow to switch one or several receivers from one stream to a new one with different parameters without having them to restart the whole reception. This is mainly useful on large multicasts where a minority of receivers have a really bad satellite reception. In
such cases, it can be advantageous to switch these receivers to terrestrial reception, where they would receive the file faster without impeding other receivers in the multicast who have a good reception. This is achieved by the use of stream switch packets.

For a switch to complete, first the sender emits a stream switch packet containing the concerned stream, the newly created stream and the receivers concerned by this switch. Then, the receivers must reply with a stream switch packet containing the same informations, except the dest field only contains its own RBC ID, to confirm the switch. Once every receivers confirm the switch to the sender (or some of them timeout), the sender starts the new stream from where the old one was switched. As shown by the figure 17, the fields of a stream switch packet are:

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>0 – 15</th>
<th>16 – 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stream id</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>new stream id</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>dest list size</td>
<td>dest 1…</td>
</tr>
<tr>
<td>96</td>
<td>…dest 1…</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>…dest 1</td>
<td>dest 2…</td>
</tr>
<tr>
<td>160</td>
<td>…dest 2…</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>…dest 2</td>
<td></td>
</tr>
</tbody>
</table>

*figure 17: Stream Switch breakdown*

**stream id (32 bits):** this field contains an ID referencing the stream to be switched.

**new stream id (32 bits):** this field contains a newly randomly generated ID for the new stream the concerned receivers will be switched to.

**dest list size (16 bits):** the size of the recipients list for the stream. As the number of recipients of a stream can vary, this field is needed to tell the receivers the size of the next field in order for them to parse it. “Dest” is a short for the french word “destinataire” meaning “recipient”.

**dest list** (variable, equals to the value of dest list size): each sender and each receiver has an RBC ID. This ID is a variable length string and is used to identify each machine using RBC. The dest list is a list of recipients containing these IDs. As RBC IDs are variable in length, each entry in the dest list is actually the first 64 bits of the MD5 hash of an RBC ID, making them fixed in length and easy to parse. In the case of a stream switch, the dest list contains the receivers concerned by the switch.
Glossary

3DES – Triple DES, Triple data Encryption Algorithm (Data Encryption Standard)
ACK – Acknowledgement
ACM – Adaptive Code Modulation
AES – Advanced Encryption Standard
ALC – Asynchronous Layered Coding
ARQ – Automatic Repeat Request
ASM – Any Source Multicast
AWGN – Additive White Gaussian Noise
BCH – Bose-Chaudhuri-Hocquenghem, inventors of this cycling error-correcting code
CC – Congestion Control
CPL – Composition Playlist
CRC – Cyclic Redundancy Check
DCI – Digital Cinema Initiative
DCP – Digital Cinema Package
DLP – Digital Light Processing
DMD – Digital Micromirror Device
DSL – Digital Subscriber Line
DVB – Digital Video Broadcasting
DVB-C – DVB for Cable
DVB-H – DVB for Hand Held Devices
DVB-S – DVB for Satellite
DVB-T – DVB for TV
FEC – Forwarding Error Correction
FLUTE – File Delivery Over Unidirectional Transport
HD – Hard Disk
IP – Internet Protocol
IPv“X” – IP version “X”
IV – Initialization vector
KDM – Key Delivery Message
LAN – Local Area Network
LCT – Layered Coding Transport
LDPC – Low-Density Parity-Check
MD5 – Message-Digest Algorithm 5
MPEG – Moving Pictures Expert Group
NAK – Negative Acknowledgement
NCF – NAK Confirmation
NORM – Nack-Oriented Reliable Multicast
OS – Operative System
P2P – Peer-to-Peer
PGM – Pragmatic General Multicast
PKL – Package List
“X”PSK – “X” Phase Shift Keying
QPSK – Quadrature Phase Shift Keying
RBC – Reliable Bit Cast
RTT – Round Trip Time
RXTX – Receiver-Transmitter (server)
SDP – Session Description Protocol
SPM – Source Path Messages
SQL – Structured Query Language
SSL – Secure Sockets Layer
SSM – Specific Source Multicast
TCP – Transmission Control Protocol
TCS – Theatre Central System
TSI – Transport Session Identifier
UDP – User Datagram Protocol
UUID – Universally Unique Identifier
VCM – Variable Code Modulation
VPN – Virtual Private Network
WAN – Wide Area Network