Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Study, design and implementation of the HD-SDI stream reconstruction module for the UltraGrid platform

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Jordi Morera Balcells

DIRECTOR: Fco. Javier Iglesias Gracia
SUPERVISOR: Xavier Miguélez Ortiz

DATA: 12 de juny de 2009

Títol: Study, design and implementation of the HD-SDI stream reconstruction module for the UltraGrid platform

Autor: Jordi Morera Balcells

Director: Fco. Javier Iglesias Gracia
Supervisor: Xavier Miguélez Ortiz

Data: 12 de juny de 2006

Resum

El treball presentat a continuació s'emmarca dins del grup de recerca i innovació sobre medià d'alta qualitat de la UPC i la Fundació i2cat.

"*Study, design and implementation of the HD-SDI stream reconstruction module for the UltraGrid platform*" té com a objectiu avaluar i dissenyar un mòdul de reconstrucció de pèrdues d'informació per una transmissió en HD (High Definition), efectuada en varis fluxos des de la plataforma de videoconferència Ultragrid.

Ultragrid és un software obert de lliure distribució i programat en llenguatge C, utilitzat per dur a terme transmissions de vídeo d'alta definició en temps real. La transmissió del flux original HD-SDI es realitza en múltiples fluxos de menor resolució que en recepció són processats per obtenir el flux original. Una particularitat destacable del sistema és que envia la informació en paquets sense comprimir-la per evitar retards en els extrems deguts als processos de compressió / descompressió. Així, s'evita perdre la tan valorada interactivitat en aquest entorn de videoconferència.

El mòdul de reconstrucció dissenyat s'ubica al receptor, i està capacitat per detectar i quantificar totes les possibles pèrdues que apareguin en qualsevol transmissió, alhora que crea nova informació de gran qualitat per tal de substituir la ja irrecuperable informació perduda. Tot això, optimitzat per no forçar els recursos del sistema i poder ser executat en temps real.

Cal afegir que aquest mòdul s'ha dissenyat partint d'un estudi profund de les possibles pèrdues i la repercussió que té cada una d'elles en la visualització del vídeo, i s'ha implementat utilitzant la interpolació de píxels com a arma principal per afrontar la reconstrucció de la informació perduda.

Al final del projecte es poden veure els resultats obtinguts per a cada reconstrucció, analitzats objectivament amb ajuda d'eines de mesura de quantificació de qualitat.

Title: Study, design and implementation of the HD-SDI stream reconstruction module for the UltraGrid platform

Autor: Jordi Morera Balcells

Director: Fco. Javier Iglesias Gracia
Supervisor: Xavier Miguélez Ortiz

Data: 12 de juny de 2006

Overview

The following project is part of the research done by the "High Quality Media Research and Innovation research group of the Fundació i2cat."

The "*Study, design and implementation of the HD-SDI stream reconstruction module for the UltraGrid platform*" has the objective of evaluating and designing a reconstruction module for losses of information on an HD (High Definition) transmission, emitted in several streams from the videoconference Ultragrid platform.

Ultragrid is an open programmed in C language, used to send high definition video transmissions in real time. The original HD-SDI transmission stream is sent in multiple low resolution flows that are processed upon reception to recover the original flow. A mentionable particularity of the system is that it sends the information uncompressed by packets in order to avoid delays in the ends due to compression/decompression processes. Thus, the loss of valued interactivity in the videoconference environment is avoided.

The designed reconstruction module is found in the receiver and has the capacity to detect and quantify all possible losses that may occur in any transmission. To solve this new high quality information is created from the well received in order to substitute the already lost information. All this is optimized to fit the systems resources at real time.

In addition, this module has been designed using as a basis a depth study on the possible losses and repercussions that each one has in the visualization of the video, and has been implemented using the interpolation of pixels as the main tool in the reconstruction of the lost information.

The results of each reconstruction can be found at the end of the project, analyzed objectively with the help of tools that measure the quality.

ÍNDEX

INTRODUCCTION

The objective of this project is to develop a reconstruction module able to solve the problems with losses that appeared in the transmission of HD data, in the Ultragrid platform.

The Ultragrid platform is an open software used as a videoconference environment. This works with real time protocol over UDP and one essential point of the development has been to code the module of reconstruction optimized to not introduce delays in the system.

This module is able to detect and save the necessary information to repair most of the possible cases of lost data, and it is done with the help of some implemented algorithms that work with pixel interpolations.

Making a resume, the project can be divided in four parts:

- The study of the different losses cases
- The different solutions developed to repair these losses
- Strategies taken for every different losses case
- The results obtained

In the initial study can be seen the classifications of all the losses that can be found in the system, how they affect it and why they appear.

In the second part are explained the detections of losses and the algorithms used in the reconstruction processes.

The strategy section shows a study of probabilities. Here is defined which are the thresholds for every type of reconstruction and here is confirmed what is the chosen solution for every losses case.

Finally, the quantitative results obtained are presented in the last part and the PSNR calculation will give objective numeric results to determinate the quality of the reconstructed images.

Besides all these, the project includes a short explanation of some important concepts, the personal conclusions extracted from the work of these last 7 month, and all the written complete C code of the developed module in the annexes.

CHAPTER 1: BASIC CONCEPTS

To understand this project is necessary introduce and define some basic concepts.

1.1. What's High Definition?

1.1.1. Resolution

High Definition standard describes different models depending on the resolution of each image and the number of frames per second (fps). These are the 1280x720 HDTV (Standard 720p) or HD Ready, the 1920x1080 HDTV (Standards 1080p and 1080i) or Full HD. The 3840x2160 SHD has become the next step on HD. Theirs progression are quadratic, in form that the format SHD (Super High Definition) and its equivalent in U.S 4k (2048x1080), represents for times the format Full HD or its equivalent 2k (2048x1080), that at the same time represents almost 4 times the size of the traditional television PAL or NTSC.[4]

In the graphic below are presented the video resolution from SD to SHD:

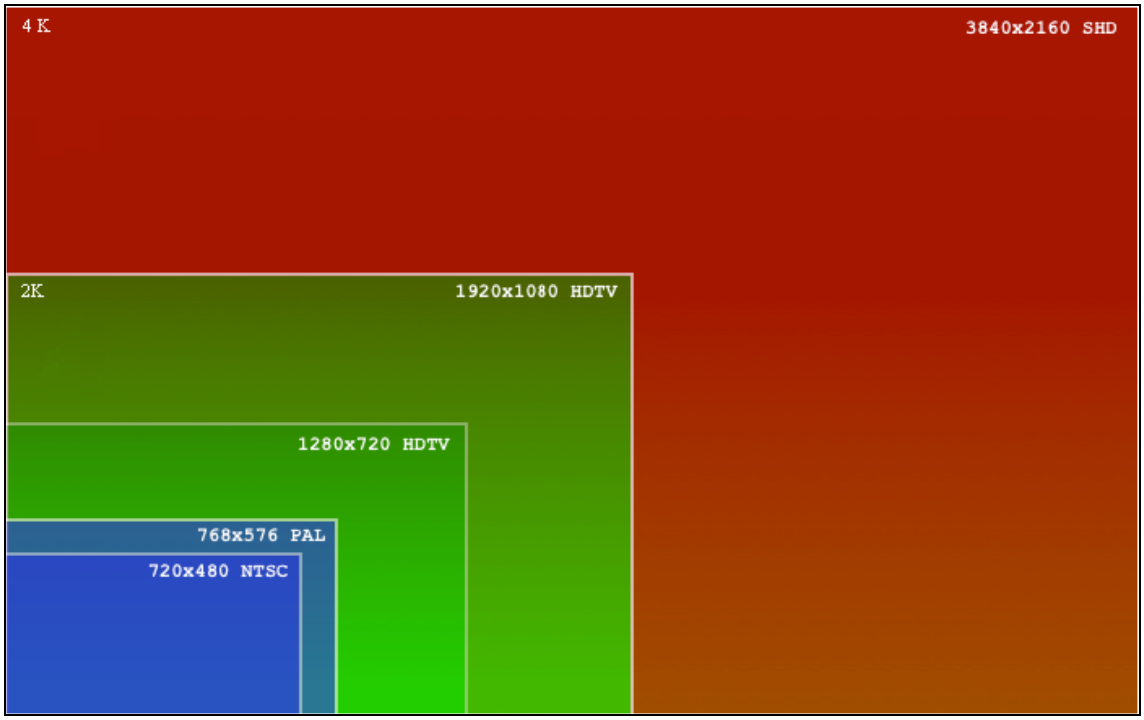


Fig. 1.1 Picture formats

1.1.2. Frame rate

Another important thing in video is the frame rate, the responsible of the receiver brain motion perception. Frame rate is the number of images displayed per second. Experimental studies prove that our visual perception starts when viewing a movement displayed around 20 fps or more. In video there are two ways of displaying images:

- *Progressive scanning (p):* where each scan displays every line in the image raster sequentially from top to bottom. Possible frame rate are 23.98 / 24 / 25 / 29,97 / 30 / 60 depending of the zone.

- *Interlaced scanning (i):* where each scan displays alternate lines in the image raster, and two complete scans are therefore required to display the entire image. Each scan is called field. Possible field rate are 50 / 59.94 / 60

1.1.3. Colour encoding

Usually RGB (Red, Green and Blue) encoding is used in electronic systems. This system assigns an equal weight to each primary colour (red, green and blue). With this encoding, it is possible to represent almost all visual colours in a black background.
Another colour space is YUV. YUV is a way of encoding RGB that offers some advantages. Firstly was thought for black and white compatibility, because YUV differs between luminance component (Y), the brightness, and the chrominance (U and V), the colour. U is the difference between blue and luminance. V is the difference between red and luminance.[4]

With the following operations is possible convert one codification RGB in YUV:

$$\begin{aligned} Y &= 0{,}299\ R + 0{,}587\ G + 0{,}114\ B \\ U &= -0{,}147\ R - 0{,}289\ G + 0{,}436\ B = 0{,}492\ (B - Y) \\ V &= 0{,}614\ R - 0{,}515\ G - 0{,}100\ B = 0{,}877\ (R - Y) \end{aligned} \qquad \textbf{(1.1)}$$

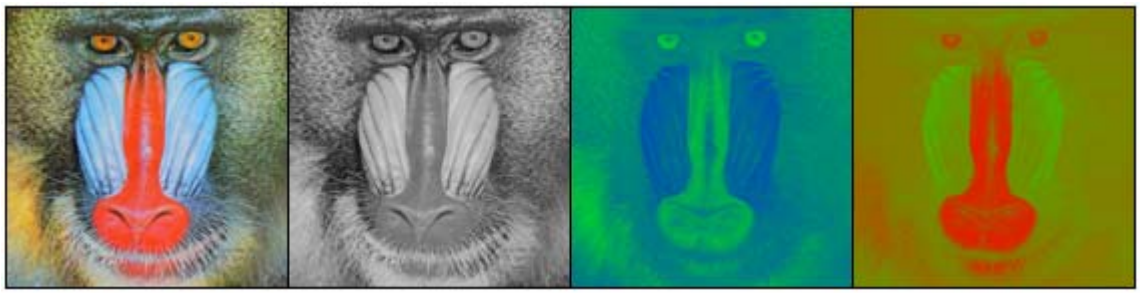


Fig. 1.2 Example of one original image and its three YUV decomposed images

Different colour encoding possibilities can be obtained using the YUV components, and these are the most typical:

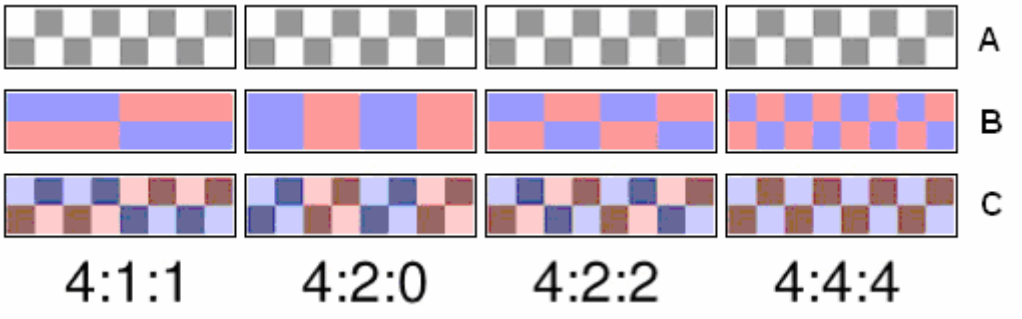


Fig. 1.3 Typical codifications with YUV components

A- Shows the luminance of all 16 pixels. Its component Y.

B- Shows the chrominance of all 16 pixels. Its U and V component. As can see in three of the four codifications, some blocks of pixels have the same colours, this wants to represent that the codification have made averages with the chrominance of these pixels.

C- Shows the result of every codification.

Since the human visual system is much more sensitive to variations in brightness than colour, a video system can be optimized by devoting more bandwidth to luma than to the colour difference components. In this scope the 4:2:2 scheme appears which requires two-thirds of the RGB bandwidth. This reduction results in almost no visual difference between RGB and YUV422 perceived by the viewer. 4:2:2 is commonly used in High Definition video, for instance any HD trailer watched in the apple website.

1.2. Ultragrid (UG)

Ultragrid (UG) is an open source application, [4] initially designed to stress the high performance networks, that allows the capture / transmission / reception of uncompressed HD (HD-SDI) over IP networks. One UG node converts High definition signals of SMPTE 292M into RTP/UDP/IP packets, to be distributed over an IP network, the video mode used in this project has the next data rate:

(1920x1080@60i ---- 4:2:2 ---- 8 bits/component)

1920 columns x 1080 lines x 30 frames/sec x 2 components/pixel x 8 bits/component = 995.32 Mbps **(1.2)**

This application has an important difference with all the other HD transmission platforms, it is designed for the Real Time transmission, trying to allow the interaction between participants. That means:

1. The buffering is reduced at the maximum level in order to reduce any unnecessary delay.
2. There is not time in compression/decompression, or possible lose of information.
3. 100 % digital cinema quality.
4. Interaction possible.

1.2.1 The functional analysis of the UG

Ultragrid can be divided in two functional parts:

Transmission side. That includes:

- The capture card module, initialization, video capture…
- The RTP layer, session establishment, packet headers...
- The transmission module, packet transmission, payload headers…

Reception side. That includes:

- The RTP layer, session establishment, checking of the header parameters…
- The decoding module, how to convert from packets to the image data…
- The capture card module, initialization, video display…

1.2.2 Transmission by flows

In order to reach different users with the same source flow, the UltraGrid software is able to send the original HD flow through several autocontent sub-streams of lower bandwidth, as a kind of MDC (Multiple Description Coding).

The original image can be divided in 4, 9, 16, 25, 36 or 64, the new images will be like the original but with less resolution. Finally in the receiver, with all the info from every flow can be reconstructed the original image or receiving some info can be presented one image smaller than the original; this project has been developed to solve the possible problems of losses that can appears in the first case.

Every sub-image is transmitted by a different RTP session. The only relation between the different sessions is the use of the same timestamp for all the sub-images related to the same original frame. With this timestamp the receiver will be able to synchronize between the sub-streams.

The figures that will be presented in the two next pages explain better all that has been said.

Fig. 1.4 Original image

Fig 1.5 Image divided in 4 flows

Fig 1.6 Image divided in 9 flows

Fig 1.7 Image divided in 16 flows

Fig 1.8 Image divided in 25 flows

Fig 1.9 Image divided in 36 flows

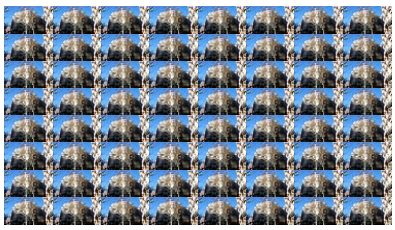


Fig 1.10 Image divided in 64 flows

The last figures show the possible divisions that Ultragrid can do. With more sub-image more resolutions can be offer to the transmitter, the next figures shows the possible composition that can be done in the receiver dividing the original image in 64.

Fig 1.11 Composed images with different number of sub-images of one original image divided in 64 parts

The division process of the original image follows a constant pattern, in the next figures an example of one division is shown:

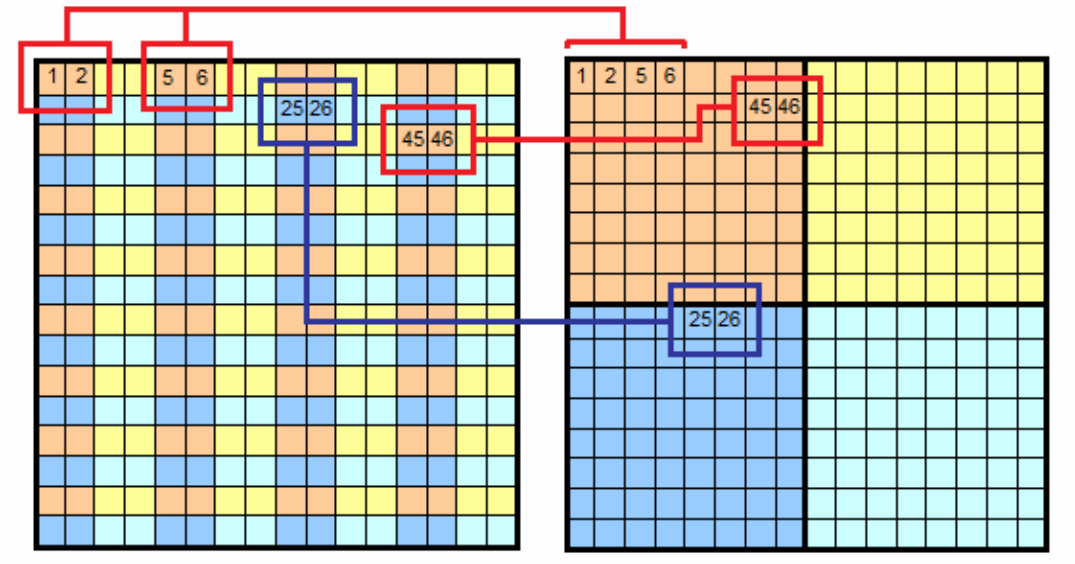


Fig. 1.12 Original image and divided image with his pixel compositions

1.3 Image reconstruction

To made reconstructions of digital images the algorithms of interpolation are the tool more used.

Conceptually, interpolation works using known data to estimate values at unknown points. Nowadays, many different interpolation algorithms exist and these are the most common:

1.3.1. Nearest neighbour interpolation

Nearest neighbour is the most basic and requires the least processing time of all the interpolation algorithms because it only considers one pixel, the closest one as the interpolated point. This has the effect of simply making each pixel bigger.

1.3.2. Bilinear interpolation

Bilinear interpolation considers the closest 2x2 neighbourhood of known pixel values surrounding the unknown pixel. It then takes a weighted average of these 4 pixels to arrive at its final interpolated value. This result is much smoother looking images than nearest neighbour.

1.3.3. Bicubic interpolation

Bicubic goes one step beyond bilinear by considering the closest 4x4 neighbourhood of known pixels-- for a total of 16 pixels. Since these are at various distances from the unknown pixel, closer pixels are given a higher weighting in the calculation. Bicubic produces noticeably sharper images than the previous two methods, and is perhaps the ideal combination of processing time and output quality. For this reason it is a standard in many image editing programs (including Adobe Photoshop), printer drivers and in-camera interpolation.

1.3.4. Spline & sinc interpolation

There are many other interpolators which take more surrounding pixels into consideration, and are thus also much more computationally intensive. These algorithms include Spline and Sinc, and retain the most image information after an interpolation. They are therefore extremely useful when the image requires multiple rotations / distortions in separate steps. However, for single-step enlargements or rotations, these higher-order algorithms provide diminishing visual improvement as processing time is increased. These algorithms have been discarded quickly for their complexity.[13]

For the project, a new algorithm of pixel interpolation optimized for the specific necessities of the system has been developed. This works similar than the bilinear / bicubic algorithms but have some characteristic points that will be shown later.

CHAPTER 2: STUDY OF THE POSSIBLE PROBLEMS FOR THE DECODING SYSTEM

2.1. First approach

In this chapter, all cases where the Ultragrid's decoder can be surpassed are presented, on a extended classification.

As in common communication systems, Ultragrid videoconference system can fail in the transmitter, in the channel (net) or in the receiver. The next table shows the origin of the losses in each case:

Table 2.1 Origin of the losses

Place of the communication	Possible problems for the future decoding generated
Transmitter	Overload
	Partially or completely cut transmission
Channel (net)	Overload
	Lost data
Receiver	Overload
	Partially or completely cut reception

Regarding to the table above, all of these problems can make to lose or reject RTP packets, and affect the visualization of the video, for instance green pixels from the lost data appear or simply in the worst case, an entire frame is not received losing the real-time, essential for this kind of system.

This is the starting point of the analysis. With the transmission of multi-resolution uncompressed HD-video, the pixel information are sent in RTP packets by different flows, and different problems could appear in the decoder because the system needs all the information that comes into the packets to paint every frame from the stream of 30 frames per second.

The study presented in this report is divided basically in two parts:

- In 2.2 what happened in the frame is shown when one or more than one packet is lost or how these losses affects to the pixel map of the current frame.
- In 2.3 different problems found when losses affect different frames along the time are presented or also how these losses affects to the pixel map of the current frame.

The 2.4 introduce something else about the losses and the flows but this will be better expanded in the next two chapters.

2.2. Problems in the spatial dimension when RTP packets are lost

In this section 2.2, it is considered that the temporal dimension doesn't exist. So, the consequences of losing RTP packets in the spatial dimension will be shown, and how the system works with the received data to recover every frame.

To describe the possible lost cases and for doing it easier for the reader, the follow tables with very intuitive images have done to show them. This kind of images will be repeated along the chapter.

Legend to understand the tables:

Every full HD frame can be divided up to 64 sub-images and the packets often contain more than one information line. The next points will be considered:

- In transmission the image is sent in four flows.
- Every packet of every flow contains only one information line of his sub-image.
- The image resolution will be of 16x16 pixels
- The lost information is painted in green.

Before transmission

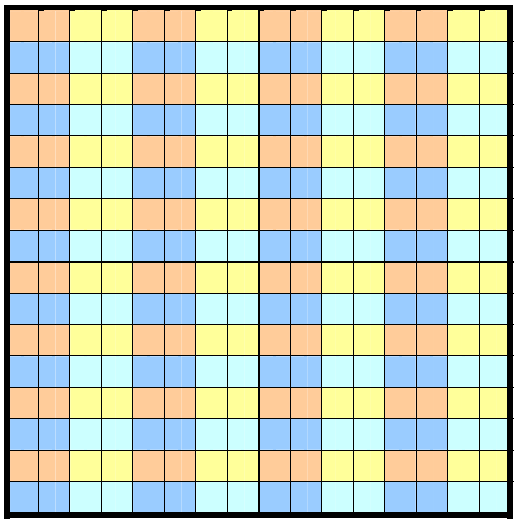


Fig. 2.1 Original image with his pixel composition

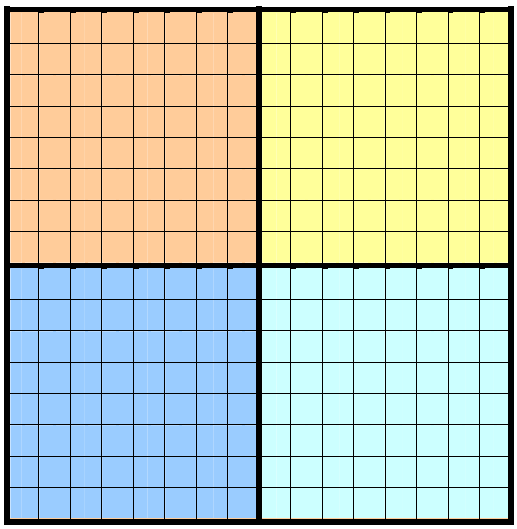


Fig. 2.2 Original image divided in four sub-images

The figure 3.1, show the pixel-map of one frame before the transmission, every little painted box represents one pixel and, as it can see, they are in pairs due to

the composition of YUV 4:2:2. The figure 3.2 shows the image divided in his four sub-images.

Summary of the next cases:

1. Losing a packet
2. Losing more than one packet
   - Packets with remote information
   - Packets with neighbour information
3. Losing all the packets

After transmission

2.2.1. Losing a packet

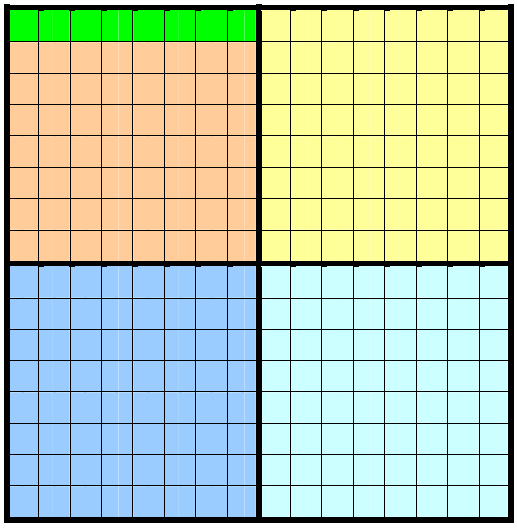


Fig. 2.3 Sub-images received, the first of them has one line lost

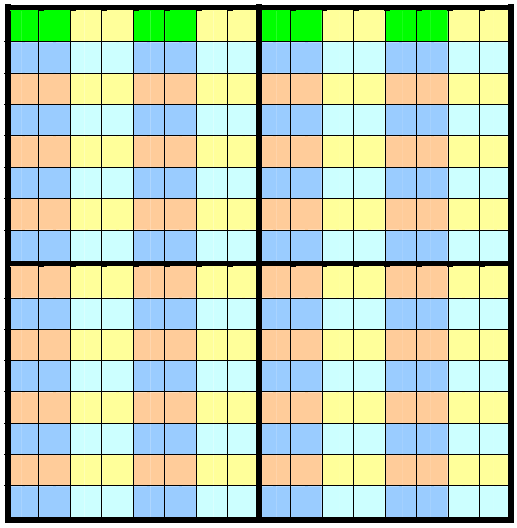


Fig. 2.4 Frame reconstructed with loses

This is the easier case. There is one lost packet and like in the reconstruction, the image is redrawn using the information of the flows, in this case, our lost pixel groups are separated.
Using an interpolation technical the losses can be repaired because the information of all surrounding pixels exists.

2.2.2. Losing more than one packet

Losing more than one packet becomes a problem more complicated. In this case has sub-cases because there are different ways to solve them.

2.2.2.1. No consecutive packets

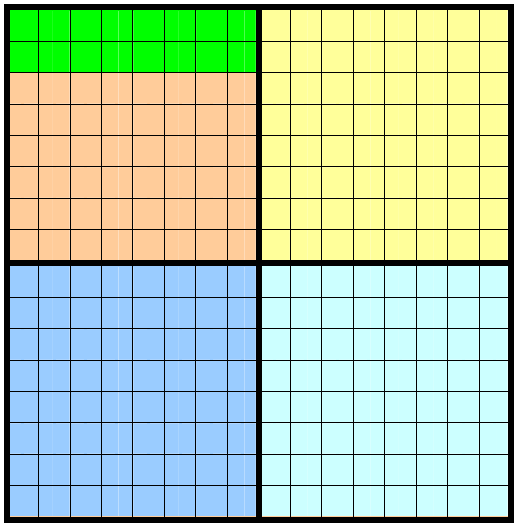


Fig. 2.5 Sub-images received, the first of them has two line lost

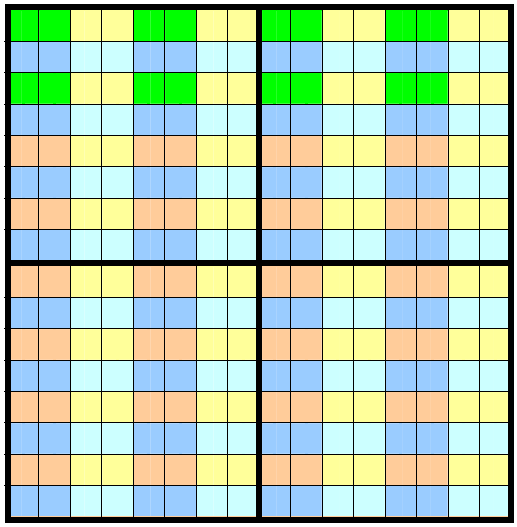


Fig. 2.6 Frame reconstructed with loses

In the last images can be seen that the lost information from the packets of every flow has not the same “Yoffset”, “Yoffset-1” or “Yoffset+1” and therefore the reconstructed image has lost information where their lost pixel lines aren’t neighbours.

2.2.2.2. Consecutive packets

The point 2.2.2.2 shows the possible complications losing more than one packet.

The sub-images have been decoded with lost information in all of them. Consequently, the reconstructed image has pairs of green pixels, lines of green pixels and blocks of green pixels. The three marks A, B and C shows the three different problems:

A- Seems similar than the 2.2.2.1, in the reconstructed image can be seen pairs of green pixels too, but they are in two consecutive lines of the reconstructed image.

B- Losing two packets of different flows with information with the same Yoffset for the reconstructed image will represent the complete lost of one line of pixels.

C- Losing four packets of different flows with the same Yoffset in the sub-images will represent the formation of green blocks of lost pixels.

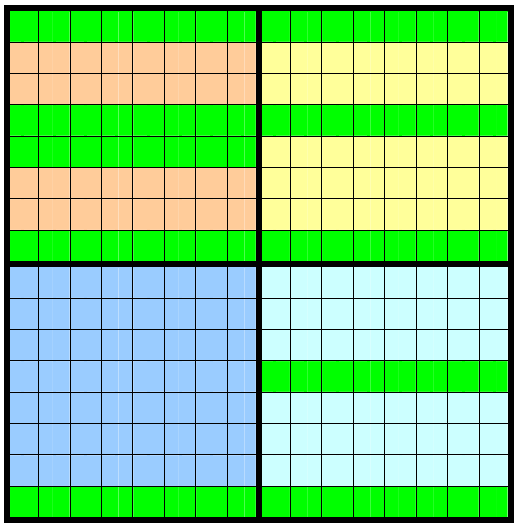


Fig. 2.7 Sub-images received with several losses

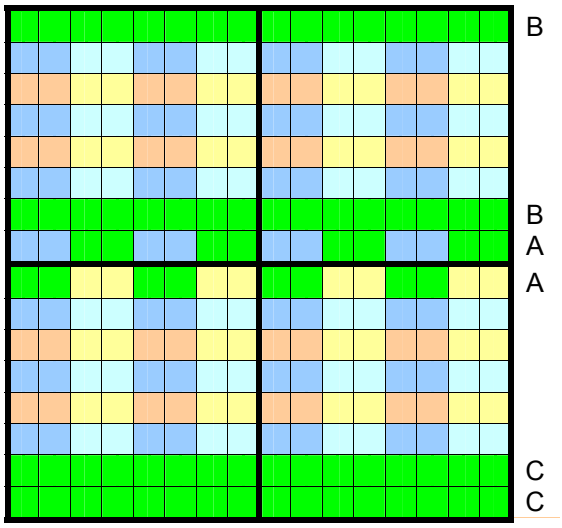


Fig. 2.8 Frame reconstructed with losses

Well, in the three cases the solution is quite complicated if the system only works on the spatial dimension, probably the interpolation techniques will not be possible because of the proximity of the losses, but we will analyze all about it in the next chapter.

2.2.3. Losing all the packets

It's the worst case and it can appear for losses but almost always appear for the overload. Nothing can be done to repair the image in this dimension.

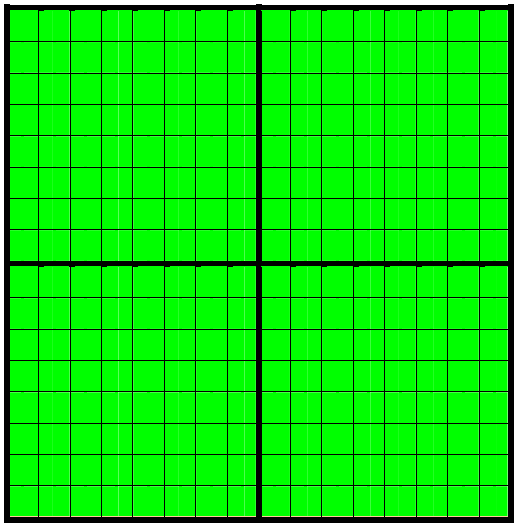


Fig. 2.9 Lost sub-images

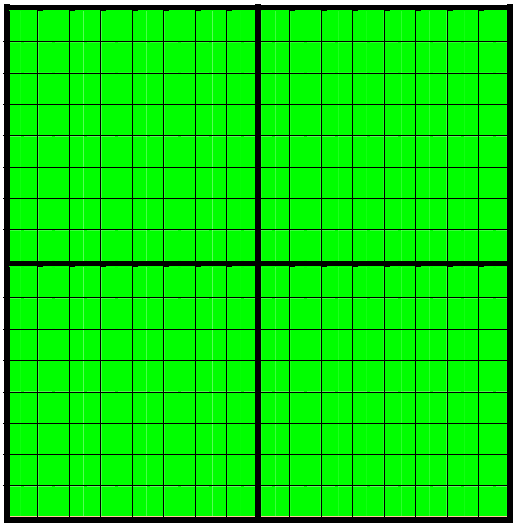


Fig. 2.10 Lost frame

2.3. Problems when we lose RTP packets in a Real-time HD video transmission.

This is the evolution of 2.1, there are data lost in the spatial dimension but furthermore, these can be extended on time or not, and even different losses can appear in consecutive frames.

It is essential to know that these will be the real problems that the system will be found. Ultragrid is designed to work in real time to make videoconferences in high definition, and the very important thing for the correct work of the system is to solve losses without losing the real time on the decoding process. These phrases are repeated many times along the project because they importance.

So far, what happened with the frames when the packets did not arrive has been studied, since here, more possibilities to play with the problems will be presented because the system will be able to use more information from the past. Nevertheless, the complexity will be higher too. In the next chapters will be see the taken strategies about it, but first, these are the considered relevant cases with the new temporal dimension in game.

The analysis follows these basic points:

- The analysis will be done using two frames: the current frame and the previous frame.
- The current frame will be the starting point.
- The previous frame will not be used if the current frame is OK.

Summary of the next cases:

1. Current frame complete
2. Current frame with partial lost information
   - Previous frame complete
   - Previous frame with different lost information
   - Previous frame with the same lost information
3. Current frame lost (with empty lost information)

After transmission

2.3.1. Current frame complete

In this case the decoding system is currently working fine. As is said, the previous frame only is need if in the current there are losses

2.3.2. Current frame with partial lost information

2.3.2.1. Previous frame complete

In the next figure can be seen the current frame with some pixels in green, this means that one packet of this image has been lost. The previous frame is complete.

This situation could be a problem with an easy solution, and the best thing is that doesn't matter the size of the losses in the current frame if there are not losses in the previous.

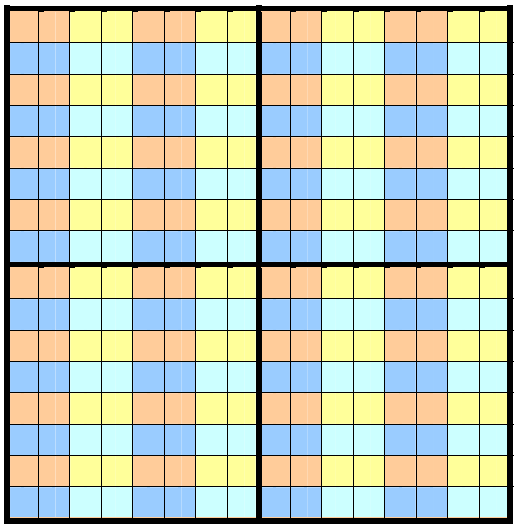


Fig. 2.11 Previous frame without losses

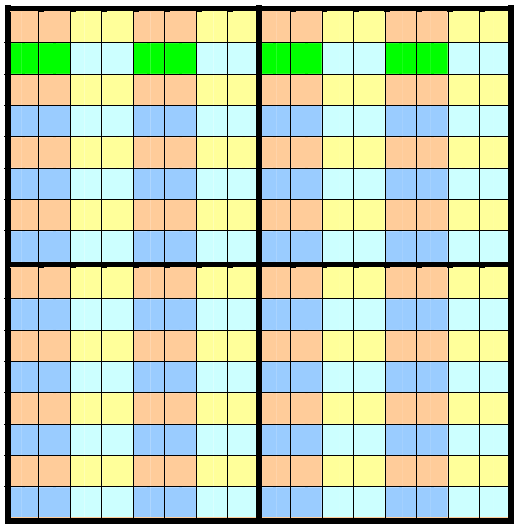


Fig. 2.12 Current frame with losses

2.3.2.2. Previous frame with different lost data

This case is very similar to the last one. The last frame is complete but the information on that is enough to be used to rebuild the current frame because their losses are not placed in the same point of the pixel-map.

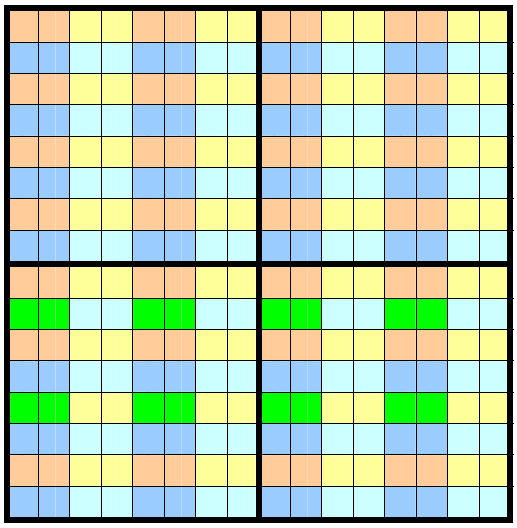


Fig. 2.13 Previous frame with losses

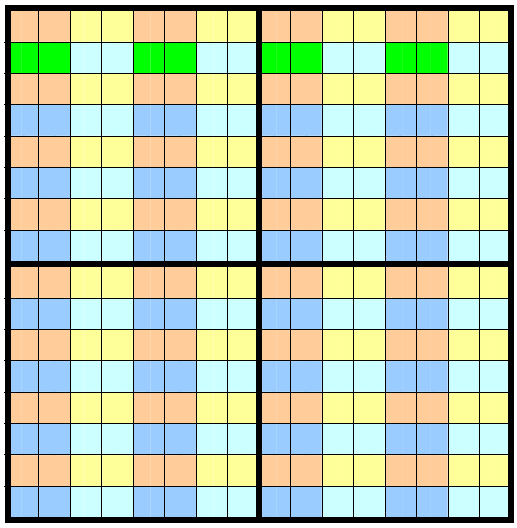


Fig. 2.14 Current frame with different losses tan previous frame

2.3.2.3. Previous frame with the same lost data

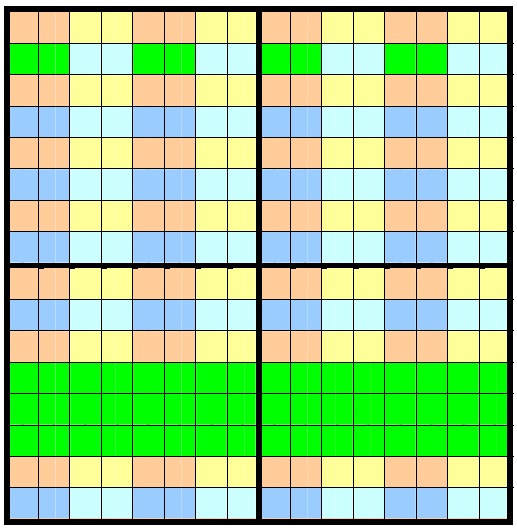


Fig. 2.15 Previous frame with the same, partially the same or the same and more losses tan current frame

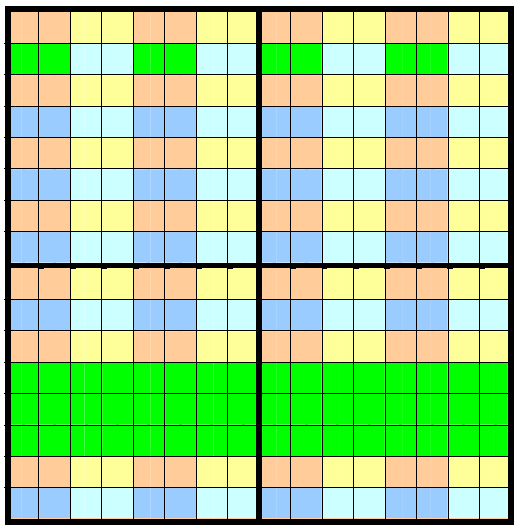


Fig. 2.16 Current frame with losses

It can be the really first case where the reconstruction of the image can not be possible as good as is want, depending on the size and the proximity of the lost pixels.

Losing the same information or partially the same information of the pixel-map in both frames, current and previous, the previous frame can be forget to help

the system for the reconstruction process. There are fewer variables to rebuild the pixel map of the current frame. This case is similar to only work with the spatial dimension seen in the point 2.1.

Having the previous frame completely lost is an extreme case and seems more complicated to solve but do not affect the system more than the others.

2.3.3. Current frame lost

If the current frame is completely lost, the solution for this problem is in the previous frame. If this also have losses and the problem persist, the system will be in serious problems and probably the communication will be cut. When one frame do not arrive, normally is because the transmitter have sent less frames to avoid the overload or the loss of packets.

2.4. Losing flows

In the project another type of losses has been considered. The losses that come from the whole flows.

The loss of one complete flow of packets is a very strange case. This can happen when there is a big overload, but in the project the case is studied because in some situations is considered that forcing these losses with the help of the transmitter, the transmission can be better with some operations more.

If there are a lot of losses but the overload has no yet appeared, the transmitter can cut one flow to reduce the losses. In the receiver will seem that the losses still exist, but they will be controlled. After this, the rebuilding process in reception can get the complete image without “green pixels” again using algorithms of reconstruction. It will be seen in the next chapters with all details.

CHAPTER 3: SOLUTIONS TO SOLVE THE DECODING PROBLEMS IN THE RECEIVER

3.1. How to detect the lost packets before the frames are presented in the screen

To detect the lost packets, an algorithm in the receiver has been implemented; concretely it is located in the decoder. The system will use it every time a packet arrives.

All received packets have travelled through the network transporting the pixel lines with their respective headers, and is here, in the headers where are found the necessary information to detect the lost packets; information like the future location (offset), the length or the flow. With this and knowing the packets arrive in order, what is the next packet that has to arrive can be predicted, and it is what the algorithm does.

Well, this is an iterative process and each iteration can be divided in four points: *declarations & initializations, data extractions from the known information, conditional comparisons and storage of information about the losses*. Then they are explained with all the specifications and a numeric example can be seen at the end.

3.1.1. Declarations & Initializations

The declarations and the initializations are the firsts of the processes and this point is the longest for making possible the understanding of the followings. Three vectors are special in the algorithm over all variables. These are their groups and their characteristics:

3.1.1.1. Vectors for saving info about arrived packets

- Two vectors control the information from incoming packets. They save the offset of the transported information and this offset is referenced to their sub-image. One vector saves the offset of the current packet and the other saves the offset of the previous.

- This is their declaration:

```
int current_pckt [ number of flows ];
int last_pckt [ number of flows ];
```

(3.1)

- Both are integer type and they have 64 positions (from 0 to 63) where each position is for one specific flow.
  (Example: If the arrived packet is from flow number 2, its info will be processed and finally saved in the position 2 of the vectors.)

- The first value given to “last_pcket” vector will be the same in all their positions. It always will be the value in bytes of the last pixel of one sub-image. It can be calculated like:

$$[2 * (\text{Nº of columns in the frame}) * (\text{Nº of lines in the frame})] / [\text{Nº of flows}] \quad \textbf{(3.2)}$$

To give value at the “last_pcket” vector before the first packet arrives is necessary for two reasons:

A- The comparison between it and “current_pcket” vector is made in every iteration after and without initialize “last_pcket” it could not be possible in the first iteration.

B- The “last_pcket” vector is also used to control the possibility of losing one full flow and for do it this operation is needed the initialization of “last_packet” vector to have a reference since the beginning of the first iteration. It will be explained in point number 3.

3.1.1.2. Vector for saving info about lost packets

- This vector saves the information about the found losses
- This is its declaration:

```
{
        int offset[1000][ number of flows];
        int len[1000][ number of flows];
        int num[ number of flows ];
        int total_num;
} lost_pckts[3];
```
(3.3)

- The vector is structured type and only has three positions (from 0 to 2). They have not relation with the flows like the last pair of vectors, and they are used in a parallel process to make more efficient this system. It will be seen in the next chapter when the strategies will be introduced.
- The positions save the following information:

Table 3.1 Information saved in “lost_pckt” positions

“lost_pcket [0]”	Both positions save information about the losses of two consecutive frames (the current and the previous). These positions behave like a switch and the information is saved every time in one different position overwriting the information.
“lost_pcket [1]”	
“lost_pcket [2]”	If there are the same losses in position 0 and 1, these losses are saved in position 2.

- These are the characteristics about the four variables defined in the structure:

Table 3.2 Information saved in the structure “lost_pckt”

offset[1000][flow]	This vector of two dimensions saves the offset in bytes of where would have to go the first lost pixel in the sub-image. In other words, every packet has pixel information, this information is from one sub-image and start at one point and ends in other. This variable offset of “lost_pcket” will show to the decoder where is this starting point. For every flow this information can be saved until X times, and it is done in the arrived packets order.
len[1000][flow]	Here the length of the lost data started in “offset [1000][flow]” is saved.
Num[flow]	“Num[flow]” saves the number of lost packets in every flow. Nevertheless, there are two particular cases where the lost packets can be found and consequently, the vector num will be approximate : 1)-If the lost packets have consecutive data lost from the same sub-image, the implemented algorithm can not determinate where one packet starts and where one packet ends. The algorithm only can determinate where the losses start and where it. Therefore if a burst of packets from the same flow does not arrive to the receiver, the system will consider that one packet is lost. Anyway, this only affects the reliability of this variable because the process of reconstruction is made correctly 2).-If the last packet of any flow is lost it can be detected and then the variable “Num[flow]” will has an error of one unit. It will be seen in point 3.

total_num	Similar than the last one variable, “total_num” has the sum of all lost packets from all the flows. It is extracted from “Num[flow]” and logically if there were errors in that vector, here there will be too.

With this ends the conceptual explanation of specifications and functions about the three vectors.

3.1.2. Data extractions from the known information

Once the three vectors are declared and initialized, all is prepared to receive the packets. They will give useful information to calculate if there are lost information. For do it, in this point the implemented algorithm have to extract the following parameters from the packets:

a- The flow number
b- The X_offset
c- The Y_offset
d- The information length

As is seen, every packet normally has more than one header to control the pixels and all headers have the last information inside but only will be necessary the information from the first header and the information length from the others (d).

When X-offset (b) and Y_offset (c) are taken from the first header the next calculations are done and after, the result it is saved in the “current_pcket [flow]” vector in the position extracted previously from the “flow number”:

- X_offset is in pixels and has to be in bytes, so is multiplied by 2
- Y_offset is in line number and also has to be in bytes, so is multiplied by the number of bytes that compose one line of the sub-image.
- The sum is possible because is done with the same units and the result gives the position in the sub-image of the first pixel of this packet in bytes

$$[2*(x\_offser) + 2*(\text{number of pixels for line in the sub-image})*(y\_offset)] \quad \textbf{(3.4)}$$

Finally, is known the flow and the offset where the pixels will be painted, the total length in bytes of all contained information in the packet will be calculated and saved in an integer variable called “total_len”. The calculation is very easy; it is a sum of the lengths of all pixel lines of the packet.

$$Total\_len = \sum (\text{pixel line lengths}) \quad \textbf{(3.5)}$$

3.1.2.1. Found problems with this point

Initially, this point created problems in the algorithm development because the sorted packets did not arrive to the receiver from every flow like it was expect. In the Ultragrid platform, the packets arrived from the last to the first, in descendent order. It was not referenced since now and for future developments of the system has to be contemplated. The next image shows it with the help of an example.

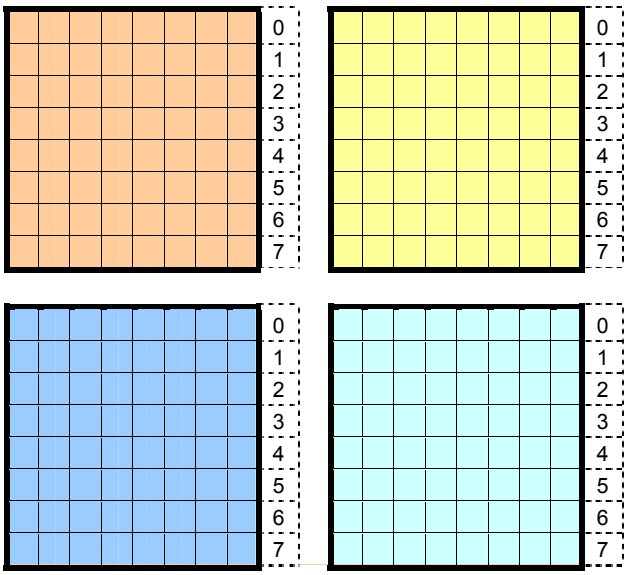


Fig. 3.1 Four flows (one for sub-images) are sent from the transmitter (1 packet-1 line)

The first arrived packet from every flow is the packet that has the lasts pixels of his sub-image, the last line. In this case the first arrived packet would have the line with number 7. And the last arrived packet of every flow will has the first line of his sub-image. In this case the line number 0 of his sub-image.

That is the reason to initialized “last_pcket” vector with the offset in bytes of the last pixel and do not initialized in zero

3.1.3. Conditional comparisons

The next comparisons are specials in the algorithm because said if there are some lost packet between the previous arrived packet and the current (in the same flow):

```
If( Current_pckt[ flow ] + total_len == Last_pcket [ flow ]){
/*There are not losses*/
}
else{
/*There are losses*/
}
```
(3.6)

The packets arrive in order from the last to the first as it is seen previously, and it enables with the last two vectors help to use this comparison efficiently. The checking has to be done in all arrived packets for controlling all the losses but there are some particular cases introduced in point number 1 that need an additional explanation:

3.1.3.1. Control the one flow lost possibility

To lose one empty flow is something strange. Cases where this is possible have been presented in the last chapter and this explanation starts assuming this.

The algorithm can detect when no packets of some flow arrive correctly to the receiver. When this happens the value of “last_pcket vector” no change in all the process, therefore comparing if this variable has the same value that the firstly assigned can known if the one flow is completely lost.

3.1.3.2. Lose the last packet of any flow

There is one case where the losses can not be detected. Looking at the last comparison can be extracted the specific problem: To detect losses the algorithm need the information between them. So if the last packet is lost, the algorithm has the information of the previous packet arrived but there is not information about the next packet of the frame because it does not exist.

Anyway, this case is contemplated but is outside the project. In the reality, the Ultragrid system’s need those packets to function correctly. As is seen in the chapter 2 the decoder knows when one flow is completely received with the help of the “M bit” which gets value “1”. This “M bit” travels in the last packet of every flow therefore do not receiving this packet the system can not determinate what arrived information could be from one frame and what from the next.

To conclude the point 3, it is important to stress that this is a differenced point because here is where the system knows for first time if the current frame information is fine or not.

3.1.4. Storage of information about the losses

That is the last point of the process. Starting from the obtained results of the last conditional, when the information is lost, some information is saved in the

structure of “lost_pckt[]”. These are the operations done by the algorithm and all of them are very intuitive:

```
Lost_pckts[].offset[][flow]=current_pckt[flow]+total_len;
Lost_pckts[].len[][flow]=last_pckt[flow]-(currentpckt[flow]+total_len);
Lost:pckts[][flow].num[flow]++;
Lost_pckts[][flow].total_num++;
```

(3.7)

This saved information will allow to know how many losses, where are them and which is their length. It was extensively explained in point number 1.

Finally, the info of current_pcket [flow] is saved in last_pckt [flow]. Every arrived packet from the frame will follows this process and the information of the lost packets will be available for its possible reconstruction

3.1.5 Example of detection of losses

It is a numerical example more visual about the last explanation. Like in other chapters the frame size used will not be HD (1920x1080) to make it easy, and is considered the packets only have one header and one information line. Finally, the transmission will be in 4 flows.

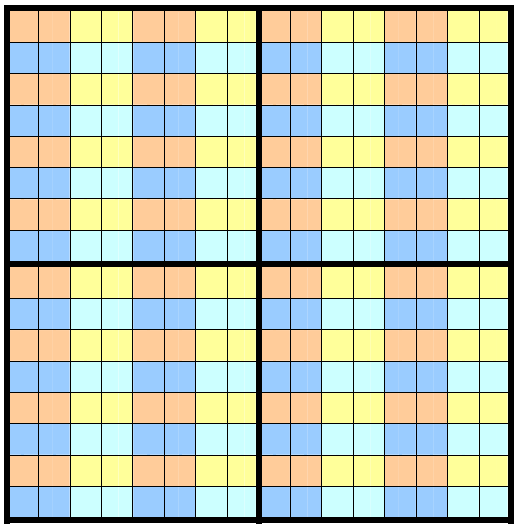


Fig. 3.2 Image to transmit

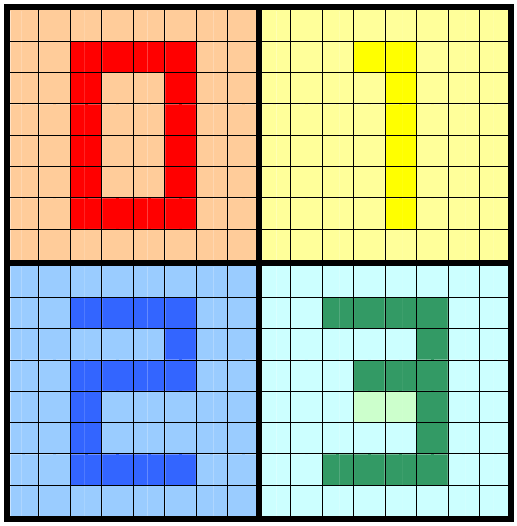


Fig. 3.3 Image divided in his sub-images to transmit

In the example can be seen three iterations of the lost detection process. The first three packets from the flow number 2 will be sent to the receiver and different problems will be analyzed.

In reception:

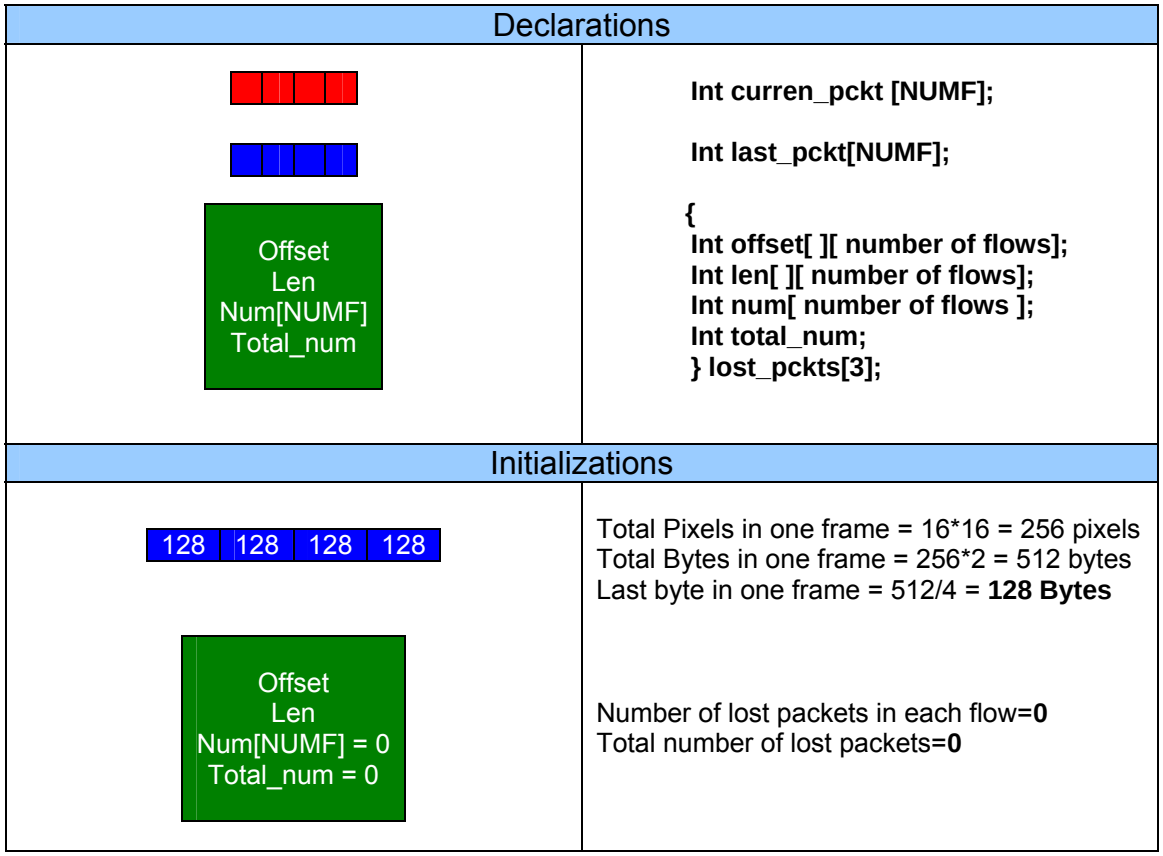
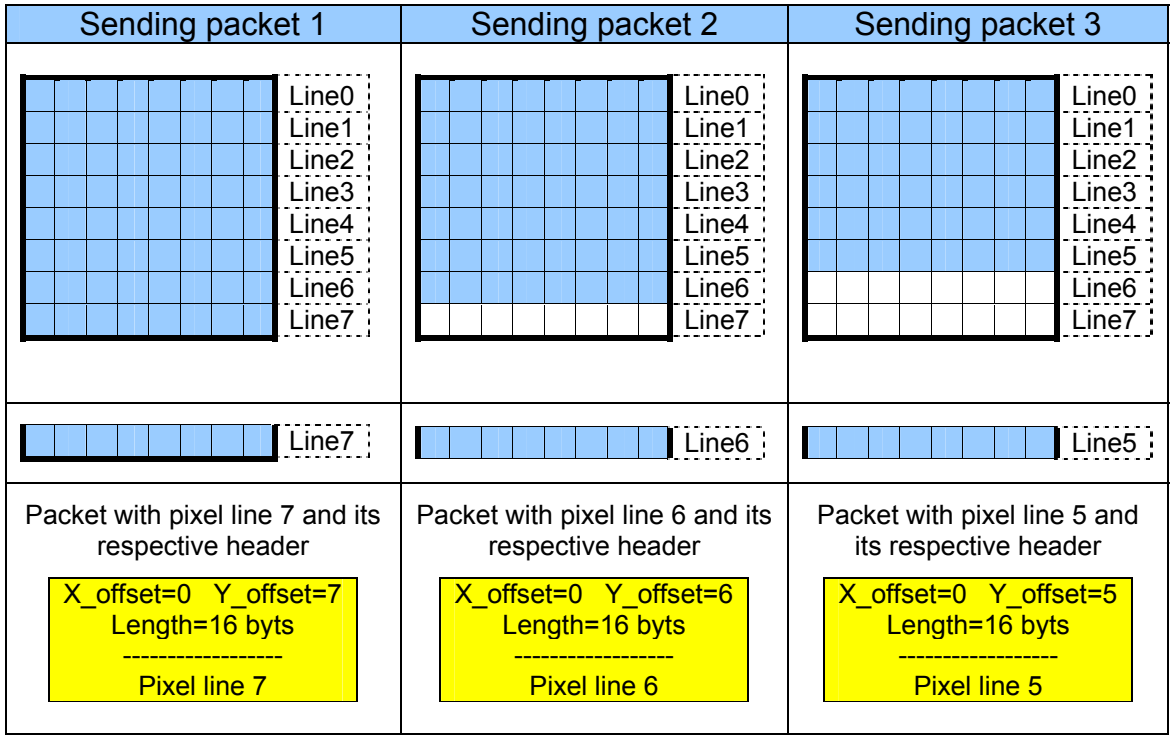


Fig. 3.4 Example of declarations and initializations in reception

In transmission:



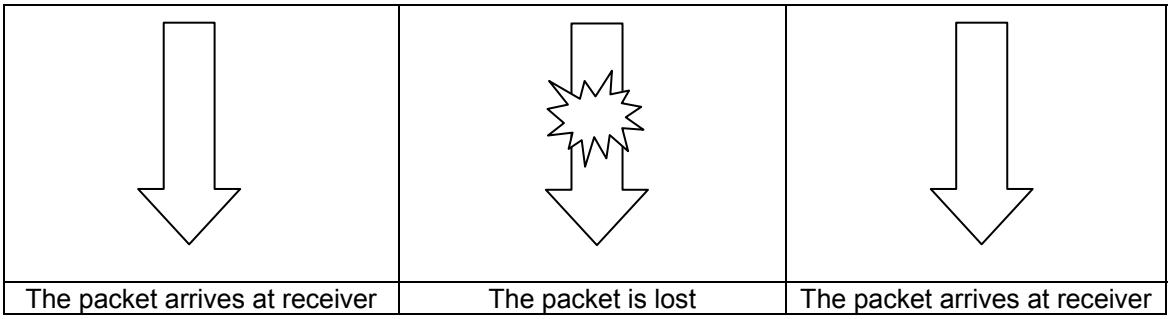
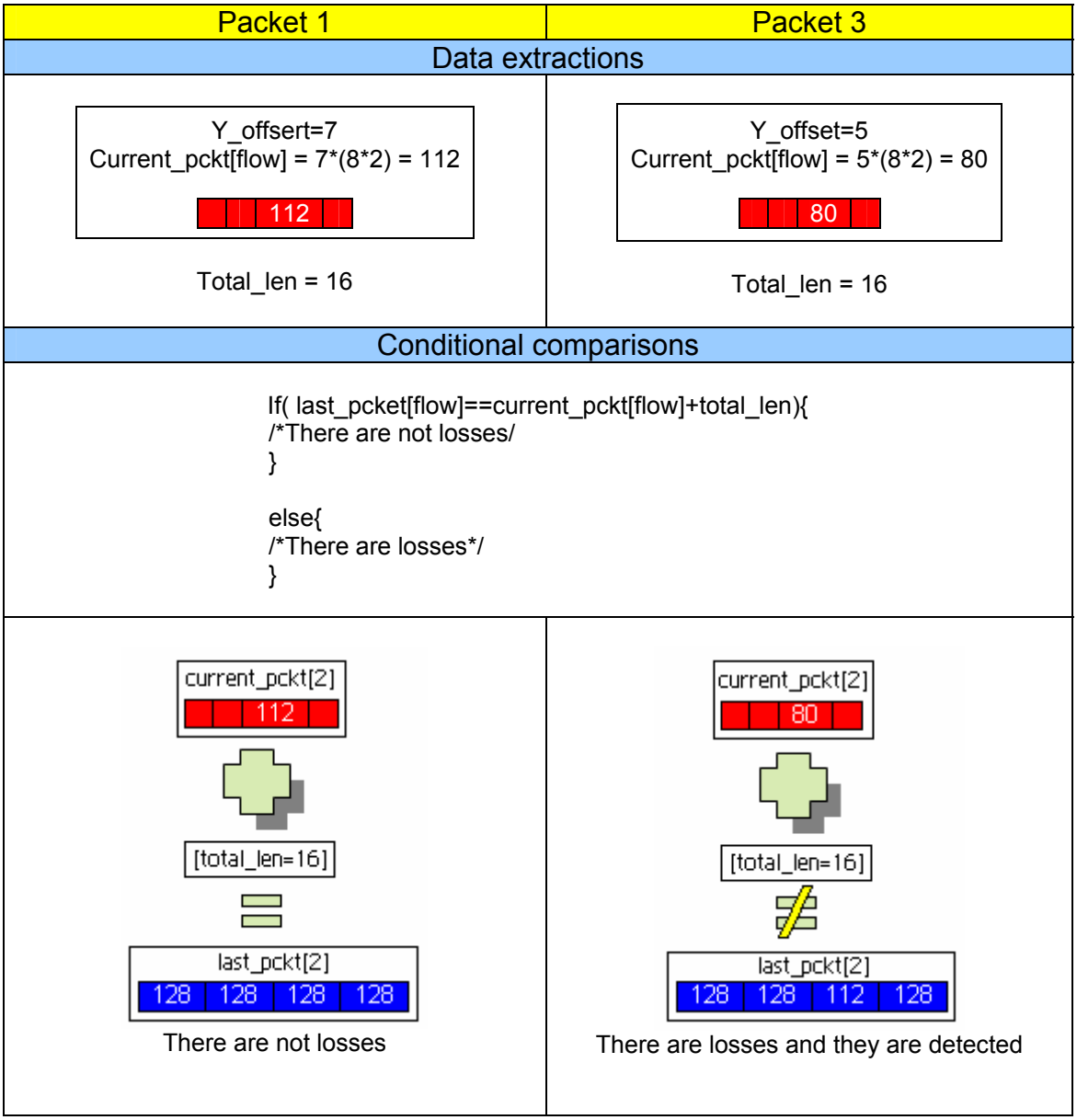


Fig. 3.5 Example of three packets sent from the transmitter

In reception:



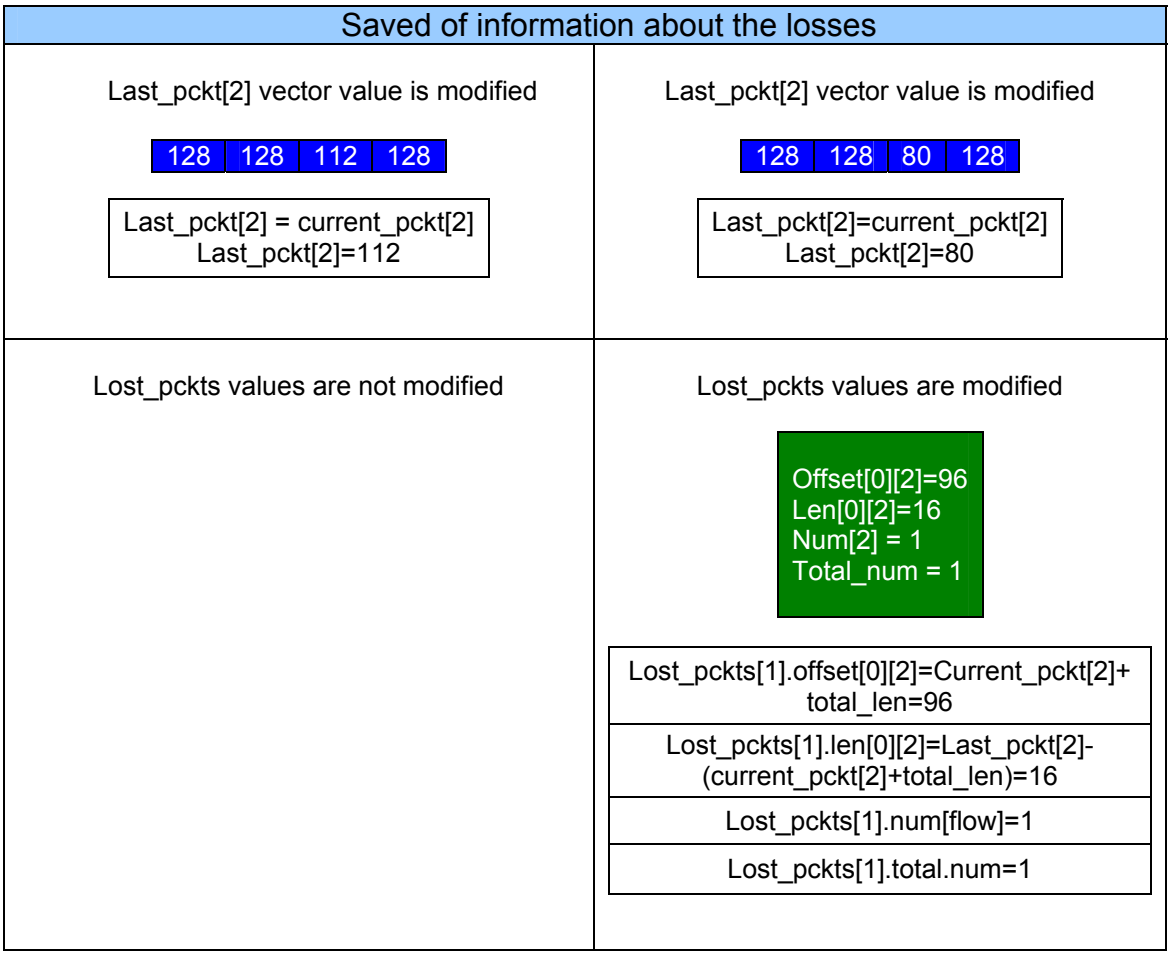


Fig. 3.6 Example of processes done in the receiver until the saved of information about the losses

3.2. Removing green pixels

What happens in the system if there are lost packets?

Once lost packets have been detected and the necessary information about them has been saved as is seen in 3.1, the system can use it to solve the problems. The process from here is the next:

- Firstly, the decoding system extracts the useful information from the received packets which was waiting in the net buffer until the last one has arrived.
- Then, the extracted information is saved orderly the screen buffer and from here this packet will be ready to be painted.
- The algorithms, presented in the previous point, work now on rebuilding the lost information before the frame is presented.
- Finally, the frame is sent to the screen without green pixels.

How to solve the problems with lost pixels?

Looking again at the study of the chapter 3, there can be seen the problems that appear when there are lost packets in the transmission. Two solutions have been studied to solve these problems and both are been explained with all details in a few lines.
The first solution is related to perform the averages between the well-arrived pixels and two algorithms have been designed with this idea. The other solution has been extracted directly from the current Ultragrid's code, therefore one is only a conceptual solution.

3.2.1. The solution of the averages (the algorithms of reconstruction)

The two implemented algorithms use the pixel interpolation to make up the frames. Really, they work with bytes not with pixels, the relation between these comes from the codification (YUV 4:2:2) used by Ultragrid. Four bytes compose two pixels and these bytes are inseparable because they form one pixel group (both pixels share the chroma). This dependence has been one of the problems for the interpolation processes because it makes the calculations more complex.

Optimization

A good optimization of the algorithms has been necessary to avoid the overload of the system because the decoder always works with high volumes of information, and with the time constrains of a videoconference environment.

With the first approximation at the existing interpolation algorithms like the nearest neighbour, the bilinear or the bicubic, something can be seen in all of them: Every algorithm is specific for a type of image and its complexities increase with its efficiency.

Therefore, the interpolation algorithms for Ultragrid's system have been developed from zero and all of them have been balanced between efficiency and complexity. The good results obtained in the realized test can be seen quantitatively in the chapter 5.

Algorithms implemented

There are two algorithms implemented to rebuild frames with lost pixels. One of them is used to rebuild frames which have lost information from some packets. This will be the most used algorithm because it can solve most of the located losses.
The other algorithm is used to solve problems of overload and it works with full flows. Its operating mode is quite different than the first but all of this will be after explained with more details.

3.2.1.1. Interpolation algorithm to rebuild losses from lost packets ("Algorithm P")

First approach

Many algorithms have been designed for rebuilding losses from packets, but only the best one has been introduced in the system, the "AlgorithmP". This algorithm can rebuild the pixel map from frames in almost all cases even though it has one limitation: if the lost pixels form a green block in the frame, or in other words, if the lost pixels have not real information around them, the "AlgorithmP" will not be able to disguise these green pixels.

Operations done in the interpolation processes

The operations done in the interpolation process are explained with a visual example; to understand it without problems the next table shows its legend:

Table 3.3 Legend for the visual example of operations done in the interpolation processes

Legend					
Colours	Number of flow	Codification	Meaning of the codification	Added sub index	Meaning of the added sub index
	Flow 0	U	Chrominance 1	f	Find
	Flow 1	V	Chrominance 2	u	Up
	Flow 2	Y1	Luminance 1	d	Down
	Flow 3	Y2	Luminance 2	l	Left
	Lost Pixels from Flow 3			r	Right

Thes next figure is the received frame. As can be seen with the legend, the second line, starting from the top, comes from the lost flow number 3. .

The lost pixels groups have real information around them; therefore this frame can be repaired with the "Algorithm P" because the needed requirements are accomplished.

To explain what the "Algorithms P" done, only one pixel group and the real information around it is needed. The discontinuous line delimits these pixels.
For the other lost pixels, the same operations are done to reconstruct them.

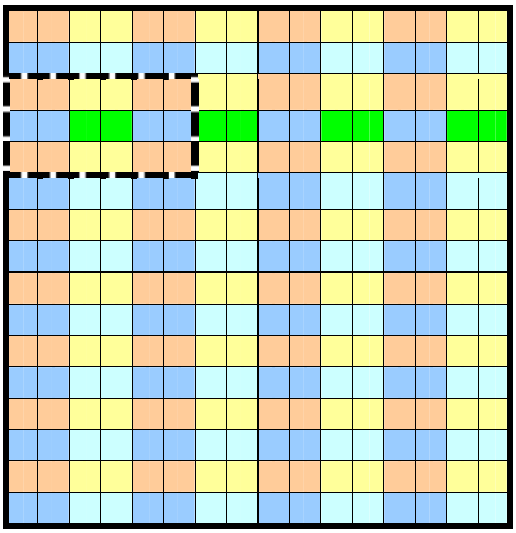


Fig. 3.7 One frame with losses from one lost packet

So, to remake the lost information the following averages has been done in the "Algorithm P":

				Uu	Y1u	Vu	Y2u				
Ul	Y1l	Vl	Y2l	Uf	Y1f	Vf	Y2f	Ur	Y1r	Vr	Y2r
				Ud	Y1d	Vd	Y2d				

Fig. 3.8 Block of pixels extracted from the last frame used to rebuild its lost pixel group.

$$\begin{aligned} Uf &= (Uu+Ud+Ul+Ur) / 4 \\ Vf &= (Vu+Vd+Vl+Vr) / 4 \\ Y1f &= (Y1u+Y1d+Y2l) / 3 \\ Y2f &= (Y2u+Y2d+Y1r) / 3 \end{aligned} \qquad \textbf{(3.8)}$$

However in cases where the lost info is in the edges or in the corners of the frame, the last averages are modified removing from the mathematical expression the non-existent bytes, except if the lost info is in the top edge, where in this case the pixels can not be reconstructed.

Following with the same arrived frame, if the last lost pixel group is taken to be reconstructed with the algorithm the modified averages are:

$$\begin{aligned} Uf &= (Uu+Ud+Ul) / 3 \\ Vf &= (Vu+Vd+Vl) / 3 \\ Y1f &= (Y1u+Y1d+Y2l) / 3 \\ Y2f &= (Y2u+Y2d) / 2 \end{aligned} \qquad \textbf{(3.9)}$$

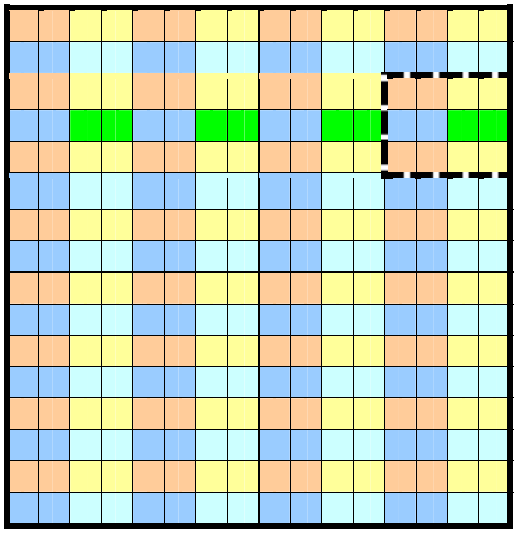


Fig. 3.9 One frame with losses from one lost packet

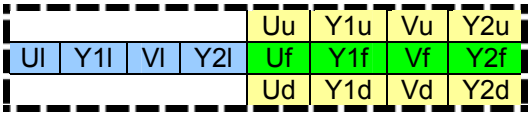


Fig. 3.10 Block of pixels extracted from the last frame used to rebuild its lost pixel group.

Other algorithms created

Before obtaining the "Algorithm P", other algorithms were created. All of them were discarded for different reasons, and are presented in the next lines:

The first to algorithms created only used the information from the previous pixel group to make the reconstruction. The results obtained were subjectively and objectively bad because the aliasing and the pixilation appeared.

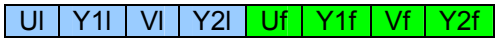


Fig. 3.11 Block of pixels extracted from the last frame used to rebuild its lost pixel group.

Two different averages have been done with this pixels to made the reconstruction:

$$\begin{aligned} Uf &= Ul \\ Vf &= V1l \\ Y1f &= Y1l \\ Y2f &= Y2l \end{aligned} \qquad \textbf{(3.10)}$$

$$\begin{aligned} Uf &= Ul \\ Vf &= V1l \\ Y1f &= (Y1l+Y2l) / 2 \\ Y2f &= (Y1l+Y2l) / 2 \end{aligned} \qquad \textbf{(3.11)}$$

This is an evolution from the previous algorithms, where the next pixel group is also used in the averages. Results are a little better but they still far from what is wanted.

Ul	Y1l	Vl	Y2l	Uf	Y1f	Vf	Y2f	Ur	Y1r	Vr	Y2r

Fig. 3.12 Block of pixels extracted from the last frame used to rebuild its lost pixel group.

$$\begin{aligned} Uf &= (Ul+Ur) / 2 \\ Vf &= (Vl+Vr) / 2 \\ Y1f &= (Y1l+Y2l) / 2 \\ Y2f &= (Y1r+Y2r) / 2 \end{aligned} \qquad \textbf{(3.12)}$$

This is the more complex algorithm implemented and it is discarded to be used because there are few differences in quality of reconstruction between this and the current algorithm used.

Uul	Y1ul	Vul	Y2ul	Uu	Y1u	Vu	Y2u	Uur	Y1ur	Vur	Y2ur
Ul	Y1l	Vl	Y2l	Uf	Y1f	Vf	Y2f	Ur	Y1r	Vr	Y2r
Udl	Y1dl	Vdl	Y2dl	Ud	Y1d	Vd	Y2d	Udr	Y1dr	Vdr	Y2dr

Fig. 3.13 Block of pixels extracted from the last frame used to rebuild its lost pixel group.

$$\begin{aligned} Uf &= (Uul+Uu+Uur+Ur+Udr+Ud+Udl+Ul) / 8 \\ Vf &= (Vul+Vu+Vur+Vr+Vdr+Vd+Vdl+Vl) / 8 \\ Y1f &= (Y2ul+Y2l+Y2dl+Y1u+Y1d) / 5 \\ Y2f &= (Y1ur+Y1r+Y1dr+Y2u+Y2d) / 5 \end{aligned} \qquad \textbf{(3.13)}$$

Better interpolation algorithms have not been developed for several reasons:

- With more complexity, the rebuilding process is slower, and it is crucial not to exceed with the overall process the threshold of 33 ms per image.
- With more pixels to make averages, a large study about which percentage is assigned to every pixel would need to be done.
- With more pixels to make the interpolation, there are more possibilities of using green pixels from other packets to make this process.

3.2.1.2. Interpolation algorithm to rebuild the losses from one full flow ("Algorithm F")

This interpolation algorithm is a derivate of the last one because this is also based on the interpolation of the eight pixels which round the lost pixel groups.

Nevertheless the “Algorithm F” has another utility. While the “Algorithm P” is used to solve problems about the losses of some packets, this algorithm solves serious problems of overload. Therefore, this algorithm will act when “Algorithm P” is exceeded.

How does the “Algorithm F” act?

It starts with one action from the transmitter. When the overload is detected, the transmitter has to send one flow less, concretely the last one. Then, the receiver can detect it (with the help of algorithm explained in 3.1 point and after, the decoding system can use the “Algorithm F” to make an interpolation with the received information to present the frame without green pixels. These are the results obtained:

- The overload should decrease.
- The obtained image will have the same size as before removing the flow
- The obtained image will be free of green pixels
- The obtained image will lose some quality because it will be made up of real and processed information.

The offsets in the “Algorithm F”

- The uncontrolled losses that comes from the net disappear and one new losses appears from the not arrived flow, this location of this last one is known by the receiver, and it facilitates the reconstruction of the pixel map.

- The flow that is cut is always the last one, if this is not respected the reconstruction will be wrong.

- The system only need to know the number of transmitted flows, with this, an offsets are calculated.

- The “Algorithm F” always makes the same operation with the pixels; the different possibilities of reconstruction are reduced to one with the help of the previous offset.

- With more flows in game more quick is the reconstruction because fewer pixels have to been reconstructed. The next table shows it:

Table 3.4 Information about the % of pixels that have to be reconstructed with the algorithm F according to the number of flows transmitted

Number of pixels by frame	Number of flows initially sent	Number of flows finally sent	Number of pixels do not sent that have to been reconstructed	% of pixels that have to been recomposed
2.073.600	4	3	518.400	25,00%
	9	8	230.400	11,11%
	16	15	129.600	6,25%
	25	24	82.944	4,00%
	36	35	57.600	2.77%
	64	63	32.400	1.56%

Example of application

This visual example shows one particular case of lost data from different flows and the solution implemented.

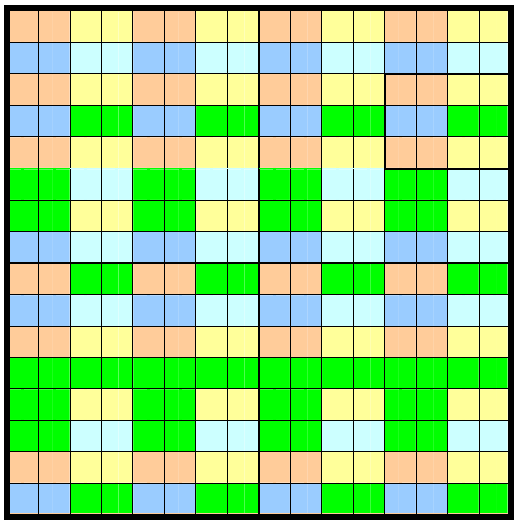


Fig. 3.14 Frame with losses from the net.

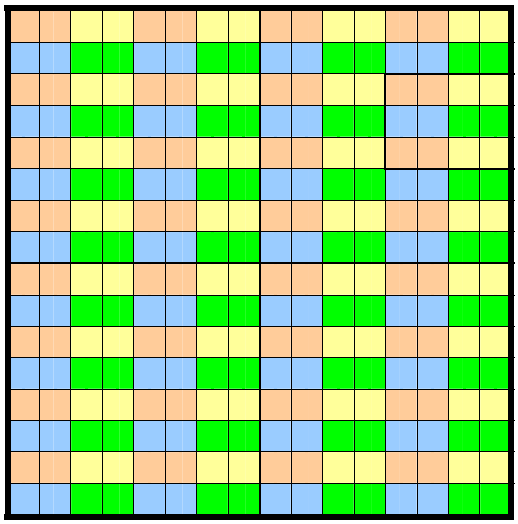


Fig. 3.15 Frame with losses fixed by the transmitter after the cut of the flow three.

The frame of the figure 3.12has a lot of lost information.Its pixel map is composed of 256 pixels and 72 are in green, this means that the 28% of the pixels are lost.With these uncontrolled losses the system will enter in an overload state in a few seconds, losing full frames and making impossible the communication.Cutting the flow 3, the transmitter will send less packets and the receiver will process less information.

The figure 3.13 is the result of cutting flow 3. The frame has losses but they are controlled because they are forced by the transmitter and the receiver knows it, and knows them location. The overflow will disappear and now, the system will

be stable. The algorithm will be able to recompose the image without problems, and finally, the frame can be seen without the green pixels

Evolution of the "AlgorithmF"

This project could be extended in the future studding the possibility of sending fewer flows or partial flows, with these ideas, the "AlgorithmF" will have to work with more variables to make the reconstruction but probably the results obtained will be better because all the process will be more controlled. The transmitter could send less information until the system was stable.

3.2.2. The solution from Ultragrid's code

It is the last solution used to solve problems caused by losses and it will be only used in some specific cases. This solution is focused on the SDL buffers (the buffers of the screen). They always have had an unknown potential to remove the green pixel and it has been taken.

How can the buffers be useful?

The system in reception has two important buffers, the incoming from the network buffer and the screen buffer. The first one is used to save incoming packets from the net. In the second one the visual useful information from the received packets is copied to be sent to the screen. This last buffer is the SDL buffer.

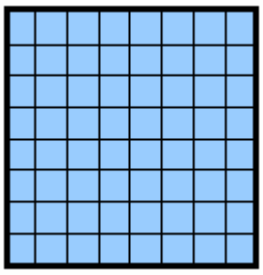
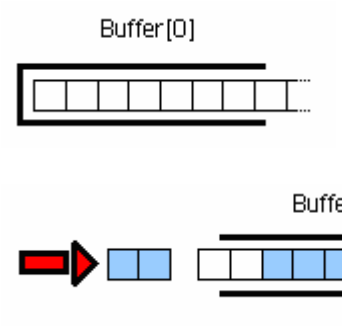
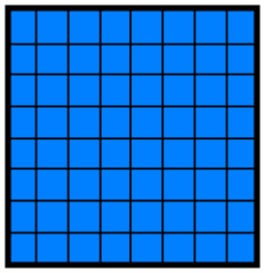
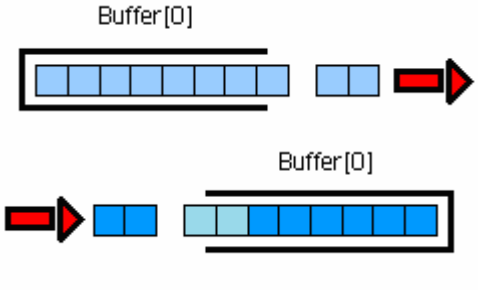
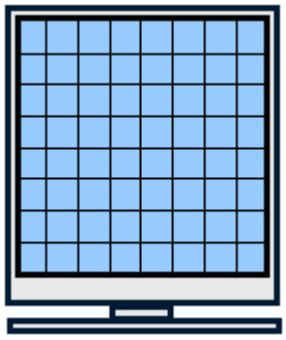
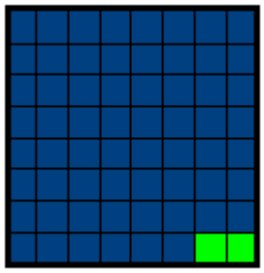
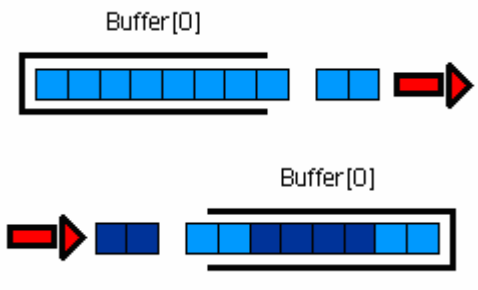
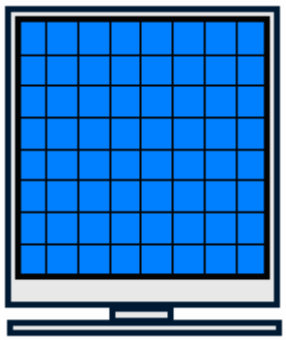
Until now in the UlltraGrid software, the SDL operated using two buffers, but in this project one of them have been deleted for the next reason:
In one SDL buffer the information was saved when it came from the net buffers. And in the other SDL buffer, the information was sent to the screen at the same time. When they finished their works both change their roles and the process started again.

The most important thing was that the SDL buffer never was deleted. So, in the next iteration, data was received from the network buffer and was saved over the last data of SDL buffer. It means that if there were losses, the data of the previous frame was still there in the buffer and could be presented. This point was very useful when consecutive frames were similar. It was a bad option in a change of scene.
The problem was in the swap of role of these two buffers. The used information from the previous frames to solve losses in the current frame was from two previous images. With only one SDL buffer the information used on losses in the current frame is the information from the last frame and the reconstruction is with pixels closer on time. The following example shows this idea:

3.2.2.1. Visual example

For this example of application is considered:

- The four first frames of one video conference are transmitted.
- The frame number 3 has lost the last 2 pixels. (This is a not realistic case because pixels travel inside packets and if one packet is lost, this one contained more than 2 pixels. Nevertheless, this makes easier the understanding of the process).

<table>
<tr><th>Transmitted frame</th><th>SDL Buffer</th><th>Shown Image</th></tr>
<tr><td> 
Frame 1</td><td> 
</td><td>Until the first iteration does not end, there is not information to be sent to the screen.</td></tr>
<tr><td colspan="3">End of time for the first iteration</td></tr>
<tr><td> 
Frame 2</td><td> 
</td><td>  </td></tr>
<tr><td colspan="3">End of time for the second iteration</td></tr>
<tr><td> 
Frame 3</td><td> 
</td><td>  </td></tr>
<tr><td colspan="3">End of time for the third iteration</td></tr>
</table>

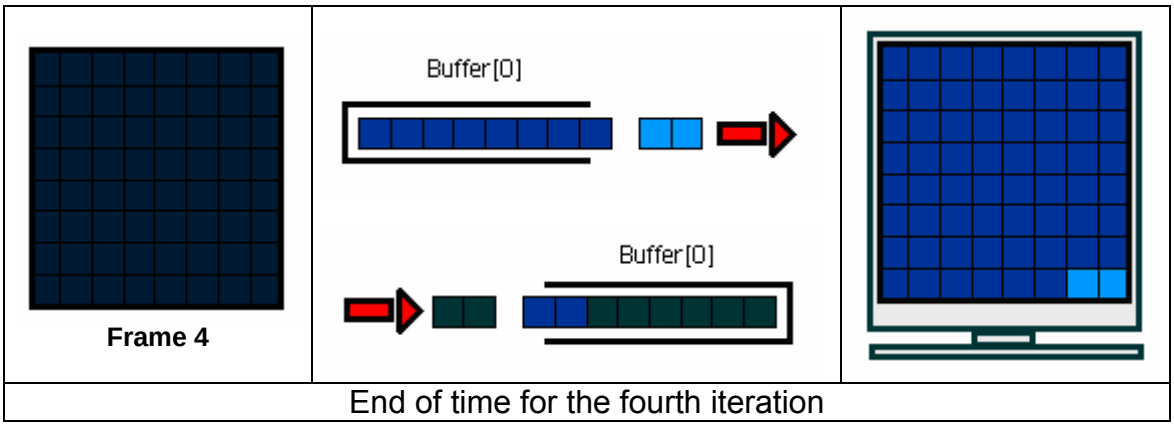


Fig 3.16 Visual example of the reconstruction using the old pixels from the previous frame

The frame number three will be presented in the screen without green pixels; furthermore, the information that has replaced those pixels is from the previous frame. Remember that this system works at 30 frames for second and the differences between consecutive frames is usually very small, therefore with high probability the losses can not be detected for human perception.

3.2.2.2 Reference files in Ultragrid's code

In main.c:

Whole process is controlled on.

In pbuf.c:

- The arrived packets wait to be worked.

In decoders.c:

- The info from the packets is saved in order in the memory address pointed by the pointer:

```
unsigned char *frame;
```
(3.14)

- The control of lost packets is implemented
- The reconstruction is implemented.

In sdl.c:

- One visualization mode is initialized
- One structure is declared as global initially. Inside this and among other variables there are the two pointers buffer:

```
struct state_sdl{
        ...
char *buffers[2];                    (3.15)
        …
        };
```

- Both buffers are initialized with size of the video that normally is 1920 px * 1080 px * 2 bytes/px:

```
buffer[0]=malloc(1920*1080*2);
buffer[0]=malloc(1920*1080*2);       (3.16)
```

- The switch between buffers is done with simply operations.

CHAPTER 4: STRATEGY

4.1. Chosen solutions

In this chapter, strategies used are defined. The next table shows the different solutions implemented in Ultragrid's system and assigns one colour to every one.

Table 4.1 Legend of implemented solutions

Implemented solutions		
"Algorithm P"	"Algorithm F"	Info from the buffers

With the support of the last table, the next one shows a classification about all the taken solutions for all the possible cases of losses that can be found:

Table 4.2 Solutions for every case of losses

Chosen solutions for the losses of one frame			
State of the previous frame / Type of losses	Without losses	With the same losses	With different losses
1 packet			
More than one packet		A B	C

4.1.1 Explanation about the chosen strategies

All solutions for the frame processed by the decoder are taken always considering what is the state of the previous frame.

Therefore, if there are losses in the current frame and:

- In the previous there are not losses, the system always will use the old info from the decoder's buffers to solve the problems. This solution is better than using the "Algorithm P" because with the buffers solution, the

results obtained are quantitative good and the system makes much fewer operations.

- In the previous there are different losses, the system also will use the same solution as before, fewer in one case:

  C- When more than one packet is lost and furthermore the overload appears, the system will use the “Algorithm F”. This is the implementation that can solve the overload.

- In the previous there are the same losses, the system will be use the “Algorithm P” to solve the great majority of cases. However, there are two conflictive cases:

  A- As it happened before, the “Algorithm F” will be used if there is overload.

  B- When losses form blocks of green pixels in the current frame, and the same losses were in the previous frame, the “Algorithm P” can not rebuild the lost info interpolating because this would use green pixels for doing it, and the results would be bad.

  The case is contemplated but the probability that it happen is almost zero, this affirmation is extracted assuming:

  - There is a low probability of losing packets if the system is not overloaded.

  - There is a low probability of losing the same packet from the previous frame if the system is not overloaded.

  - Frames are sent by several flows made up of sub-image (see chapter 1). Lost packets, that would make possible the problem, have to transport neighbour information of the complete image, and only 3 packets from 3 different flows can carry out this specification. So, the bursts have not to affect the system because they often appear in the singular RTP transmissions (in singular flow).

  - There is a low probability of losing some of the other packets that have the neighbour pixels from one lost packet.

  - There is a low probability of losing the same packets in the next frame according to all previous points.

4.2. Probability calculations

Until now, all strategies have been explained in words, this point pretends to quantify the “why of all the taken decisions”. The probability calculations about the possible losses are the best way to show it.

4.2.1. Necessary information

Resolution: (1920 x 1080)
Nº Bytes / Frame: (1920 x 1080) x 2 = 4.147.200

MTU Ultragrid: 8800 bytes
Nº Bytes of data / Packet = 8800 -8(UDP) -12(RTP) = 8780 bytes.

Table 4.2 Information about number of transmitted packets

Nº Flows	Bytes / Flow	Packets / Flow	Packets / Frame
4	1.036.800	1.036.800 / 8780 ≈ 119	119 x 4 ≈ 476
9	460.800	460.800 / 8780 ≈ 53	53 x 9 ≈ 477
16	259.200	259.200 / 8780 ≈ 30	30 x 16 ≈ 480
25	165.888	165.888 / 8780 ≈ 19	19 x 25 ≈ 475
36	115.200	115.200 / 8780 ≈ 14	14 x 36 ≈ 504
64	64.800	64.800 / 8780 ≈ 8	8 x 64 ≈ 512

Average of Packets / Frame = 487.
If the transmission is at 30 frames / second, the total number of packets transmitted are 487 x 30 = 14.610 packets / second.

4.2.2. Probabilities of losses

The results from the probability calculations can be classify in two groups: (a) the results obtained with one transmission done with 4 flows and (b) the results obtained with one transmission done with more than 4 flows.

They change because of the precedence of the surrounded pixels from every pixel group:

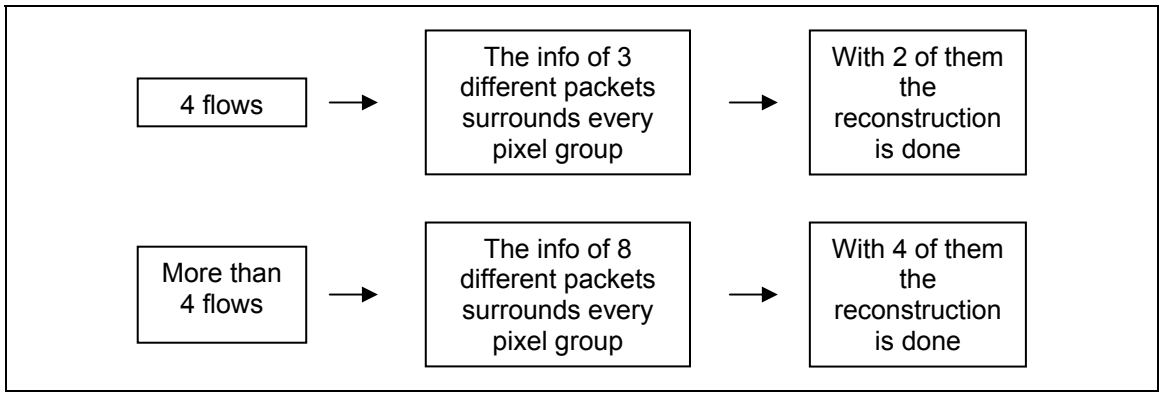


Fig. 4.1 Diagram of the differences of sending 4 frames or more.

The followings painted blocks of pixels show the different precedence from the pixels groups in the two explained cases:

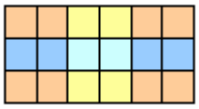


Fig. 4.2 Block of 18 pixels formed by information from 4 packets

Fig. 4.3 Block of 18 pixels formed by the information from 9 packets

These calculations of probabilities have been done assuming two different thresholds of losses that the system have to support: 5% and 10%.

4.2.2.1. Assuming the system support 5% of losses

The threshold of losses where the system will can work with out restrictions will be:

5% of 487 (the packets transmitted by frame) = 24,35 ≈ 25 lost packets/frame
5% of 14.610 (the packets transmitted by second) = 730,5 ≈
731 lost packets/seconds

Under this threshold the system will use the “Algorithm P” and the info from the buffers to repair losses. When losses will exceed this threshold, the overload of the system will be assumed and the transmitter will have to use the “Algorithm F” to solve this problem or get down the transmission rate.

(a): Probabilities of losses in the current frame for 4 flows (Locking one frame)

$$P(\text{lose one packet}) = 0.05 \qquad \textbf{(4.1)}$$

$$P(\text{lose nº packets}) = P(\text{lose one packet})^n = 0.05^n \quad \textbf{(4.2)}$$

$$P(\text{lose two packets with conflictive neighbour information}) = P(\text{lose one packet}) \times P(\text{lose some of the 2 packets with the conflictive neighbour infromation}) = (0.05)x(0.05 + 0.05 - (0.05x0.05)) = 0.05x0.0975 = 0.004875 \quad \textbf{(4.3)}$$

$$P(\text{lose three packets with conflictive neighbour information}) = P(\text{lose one packet})xP(\text{lose the 2 packets with the conflictive neighbour information}) = (0.05)x(0.05)^2 = 0.000125 \quad \textbf{(4.4)}$$

(a): Probabilities of losses in the current frame for 4 flows (Locking two consecutive frames)

$$P(\text{lose the same packets}) = P(\text{lose one packet})^2 = 0.05^2 = 0.0025 \quad \textbf{(4.5)}$$

$$P(\text{lose the same nº packets}) = (P(\text{lose one packet})^n)^2 = 0.05^{2n} \quad \textbf{(4.6)}$$

$$P(\text{lose the same two packets with conflictive neighbour information}) = (P(\text{lose one packet}) \times P(\text{lose some of the 2 packets with the conflictive neighbour infromation}))^2 = ((0.05)x(0.05 + 0.05 - (0.05x0.05)))^2 = (0.05x0.0975)^2 = 0.004875^2 = 0.000023765 \quad \textbf{(4.7)}$$

$$P(\text{lose the same three packets with conflictive neighbour information}) = (P(\text{lose one packet})xP(\text{lose the 2 packets with the conflictive neighbour information}))^2 = ((0.05)x(0.05)^2)^2 = 0.000125^2 = 0.0000000015 \quad \textbf{(4.8)}$$

(b): Probabilities of losses in the current frame for more than 4 flows (Locking one frame)

$$P(\text{lose one packet}) = 0.05 \quad \textbf{(4.9)}$$

$$P(\text{lose nº packets}) = P(\text{lose one packet})^n = 0.05^n \quad \textbf{(4.10)}$$

$$P(\text{lose two packets with conflictive neighbour information}) = P(\text{lose one packet}) \times P(\text{lose some of the 4 packets with the conflictive neighbour infromation}) = (0.05)x(0.05 + 0.05 + 0.05 + 0.05 - (0.05x0.05x0.05x0.05)) = 0.05x0.19999375 = 0.009999678 \quad \textbf{(4.11)}$$

P(lose three packets with conflictive neighbour information) = P(lose one packet)xP(lose the 4 packets with the conflictive neighbour information) = $(0.05)x(0.05)^4 = 0.000000312$ **(4.12)**

(b): Probabilities of losses in the current frame for more than 4 flows (Locking two consecutive frames)

$$P(\text{lose the same packets}) = P(\text{lose one packet})^2 = 0.05^2 = 0.0025 \quad \textbf{(4.13)}$$

$$P(\text{lose the same nº packets}) = (P(\text{lose one packet})^n)^2 = 0.05^{2n} \quad \textbf{(4.14)}$$

P(lose the same two packets with conflictive neighbour information) = $(P(\text{lose one packet}) \times P(\text{lose some of the 4 packets with the conflictive neighbour infromation}))^2 = ((0.05)x(0.05 + 0.05 + 0.05 + 0.05 - (0.05x 0.05x0.05x0.05))^2 = (0.05x0.1999375)^2 = 0.009999678^2 = 0.000099993$ **(4.15)**

P(lose the same three packets with conflictive neighbour information) = $(P(\text{lose one packet})xP(\text{lose the 4 packets with the conflictive neighbour information}))^2 = ((0.05)x(0.05)^4)^2 = 0.000000312^2 = 9.765625x10^{-14}$ **(4.16)**

4.2.2.2. Assuming the system support 10% of losses

The threshold of losses when the system will can work with out restrictions will be:

10% of 487 (the packets transmitted by frame) = 48,7 ≈ 49 lost packets/frame
10% of 14.610 (the packets transmitted by second) =
1461 lost packets/seconds

Under this threshold the system will use the “Algorithm P” and the info from the buffers to repair the losses. When the losses will exceed this threshold, the overload of the system will be assumed and the transmitter will have to use the “Algorithm F” to solve this problem or get down the transmission rate.

(a): Probabilities of losses in the current frame for 4 flows (Locking one frame)

$$P(\text{lose one packet}) = 0.1 \quad \textbf{(4.17)}$$

$$P(\text{lose nº packets}) = P(\text{lose one packet})^n = 0.1^n \quad \textbf{(4.18)}$$

P(lose two packets with conflictive neighbour information) = P(lose one packet) x P(lose some of the 2 packets with the conflictive neighbour infromation) = (0.1)x(0.1+ 0.1- (0.1x0.1)) = 0.1x0.19 = 0.019 **(4.19)**

P(lose three packets with conflictive neighbour information) = P(lose one packet)xP(lose the 2 packets with the conflictive neighbour information) = (0.1)x$(0.1)^2$ = 0.001 **(4.20)**

(a): Probabilities of losses in the current frame for 4 flows (Locking two consecutive frames)

P(lose the same packets) = P(lose one packet)^2 = 0.1^2 = 0.01 **(4.21)**

P(lose the same nº packets) = (P(lose one packet)^n)^2 = 0.1^{2n} **(4.22)**

P(lose the same two packets with conflictive neighbour information) = (P(lose one packet) x P(lose some of the 2 packets with the conflictive neighbour infromation))^2 = ((0.1)x(0.1+ 0.1- (0.1x0.1)))^2 = $(0.1x0.19)^2$ = 0.019^2 = 0.000361 **(4.23)**

P(lose the same three packets with conflictive neighbour information) = (P(lose one packet)xP(lose the 2 packets with the conflictive neighbour information))^2 = ((0.1)x$(0.1)^2$)^2 = 0.001^2 = 0.000001 **(4.24)**

(b): Probabilities of losses in the current frame for more than 4 flows (Locking one frame)

P(lose one packet) = 0.1 **(4.25)**

P(lose nº packets) = P(lose one packet)^n = 0.1^n **(4.26)**

P(lose two packets with conflictive neighbour information) = P(lose one packet) x P(lose some of the 4 packets with the conflictive neighbour infromation) = (0.1)x(0.1+ 0.1+ 0.1+ 0.1 -(0.1x0.1x0.1x0.1)) = 0.1x0.3999 = 0.03999 **(4.27)**

P(lose three packets with conflictive neighbour information) = P(lose one packet)xP(lose the 4 packets with the conflictive neighbour information) = (0.1)x$(0.1)^4$ = 0.00001 **(4.28)**

(b): Probabilities of losses in the current frame for more than 4 flows (Locking two consecutive frames)

$$P(\text{lose the same packets}) = P(\text{lose one packet})^2 = 0.1^2 = 0.01 \quad \textbf{(4.29)}$$

$$P(\text{lose the same nº packets}) = (P(\text{lose one packet})^n)^2 = 0.1^{2n} \quad \textbf{(4.30)}$$

$$P(\text{lose the same two packets with conflictive neighbour information}) = (P(\text{lose one packet}) \times P(\text{lose some of the 4 packets with the conflictive neighbour infromation}))^2 = ((0.1)x(0.1+0.1+0.1+0.1-(0.1x0.1x0.1x0.1))^2 = (0.1x0.3999)^2 = 0.03999^2 = 0.0015992 \quad \textbf{(4.31)}$$

$$P(\text{lose the same three packets with conflictive neighbour information}) = (P(\text{lose one packet}) x P(\text{lose the 4 packets with the conflictive neighbour information}))^2 = ((0.1)x(0.1)^4)^2 = 0.00001^2 = 1x10^{-10} \quad \textbf{(4.32)}$$

4.2.3. Resume of probability calculations

Table 4.3 Probabilities calculated

Probabilities of the different losses in %				
Probabilities	Number of flows sent			
	4 flows		More than 4 flows	
	Percentages of losses assumed			
	5%	10%	5%	10%
Looking one frame				
P(lose one packet)	5%	10%	5%	10%
P(lose two packets with conflictive neighbour information)	0.4875%	1.9%	0.999967	3.99%
P(lose all packets with conflictive neighbour information)	0.0125%	0.1%	0.0000312	0.001%
Looking two consecutive frames				
P(lose the same packets)	0.25%	1%	0.25%	1%

P(lose the same two packets with conflictive neighbour information)	0.0023765%	0.0361%	0.0099993	0.15992%
P(lose all the same packets with conflictive neighbour information)	0.00000015%	0.0001%	0.000000 0000097%	0.00000001%

As it can see, the probabilities of having losses are very low assuming a threshold of 5% and they increase considerably with a threshold of 10%. Both cases are studied for showing the evolution of losses, and these values are taken arbitrarily thinking that the system does not have to work with high volume of losses.
So, all the losses that exceed the 10% of the transmitted packets will be considered as overload.

CHAPTER 5: RESULTS

To rate the quality of different reconstructions the PSNR has been used as an objective tool.

5.1. The PSNR

The phrase peak signal-to-noise ratio, often abbreviated PSNR, is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

As it is said, in the project the PSNR has been used as a objective measure of quality of reconstruction. The signal in this case is the original data, and the noise is the error introduced by the losses.
When comparing the different images, it will be used as an **approximation** to human perception of reconstruction quality. Therefore in some cases one reconstruction may appear to be closer to the original than another, even though it has a lower PSNR (a higher PSNR would normally indicate that the reconstruction is of higher quality).

It is most easily defined via the mean squared error (**MSE)** defined as:

$$MSE = \frac{1}{M \times N} \sum_{y=0}^{(M-1)} \sum_{x=0}^{(N-1)} \left[s(x,y) - s_t(x,y) \right]^2 \qquad \textbf{(5.1)}$$

Where MxN are the dimensions of the image, s(x,y) is the original signal (the original image) and $s_t(x,y)$ is the reconstructed signal (the reconstructed image).

The PSNR is defined as:

$$PSNR(dB) = 20 \times \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \qquad \textbf{(5.2)}$$

Here, MAX_i is the maximum possible pixel value of the image.

$$MAX_I = 2^8 - 1 = 255 \qquad \textbf{(5.3)}$$

The range of values that can be obtained from the PSNR and their translation to quality of reconstruction is:

Table 5.1 Relation between values of PSNR and human perception

20 dB	Bad result
30 dB	Good result
40 dB	Excellent result

Finally, from the formula of the MSE can be extracted that when two images are identical this will be equal to zero, resulting in an infinite PSNR. [10][11]

5.2. Results obtained

As was explained in 5.1 the PSNR gives an objective reference of quality of image reconstruction. In 5.2 can be seen these results for the two developed algorithms: the “Algorithm P” and the “Algorithm F”.

There are not results of reconstruction quality from SDL buffers because would be need to capture two consecutive frames of one video transmitted in real time. With these requirements is better to evaluate the technique of reconstruction in the project presentation with one real transmission.

For extracting subjective results, images with losses and their respective reconstructions have been added besides the numerical values.

5.2.1. Results obtained with the “Algorithm P”

5.2.1.1. When 1 packet is lost

Fig 5.1 Image with one lost packet and its reconstruction with the “Algorithm P”

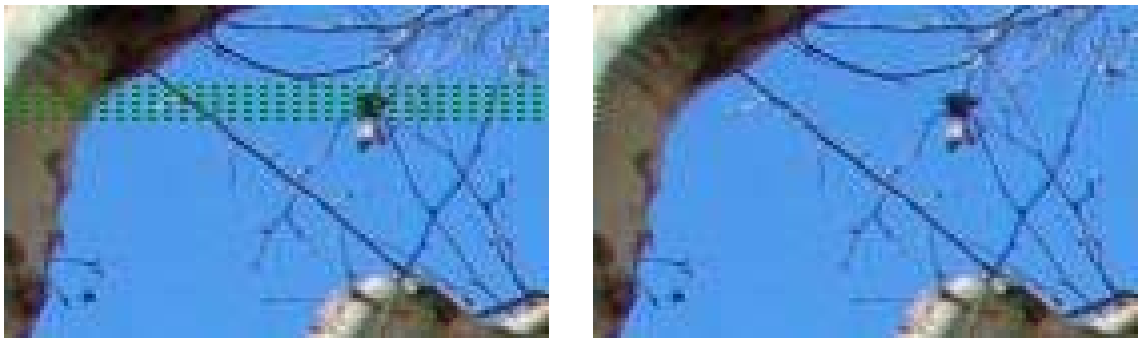


Fig 5.2 Zooms of one part of the last two images

Results of the comparison between the original image and the image with losses:

MSE=62 **(5.4)**

PSNR=30.20 dB **(5.5)**

Results of the comparison between the original image and the reconstructed image:

MSE=55 **(5.6)**

PSNR=30.72 dB **(5.7)**

5.2.1.2. When 1 burst of 25 packets is lost

Fig 5.3 Image with one burst of 25 lost packet and its reconstruction with the "Algorithm P"

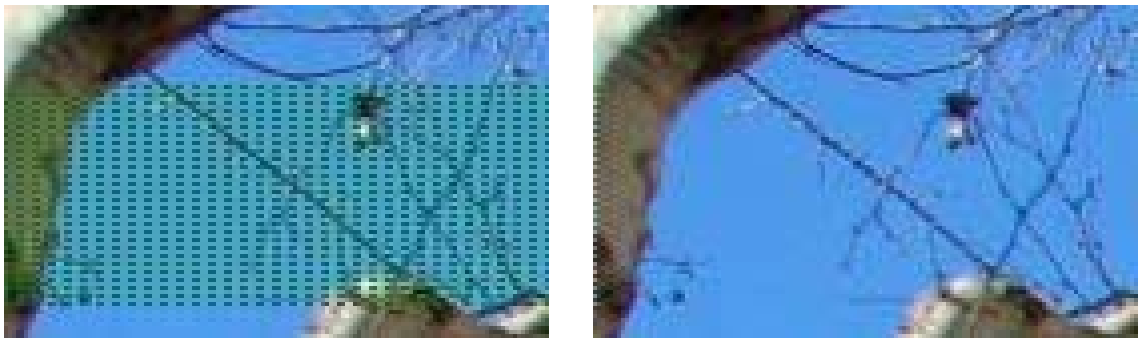


Fig 5.4 Zooms of one part of the last two images

Results of the comparison between the original image and the image with losses:

MSE=303 **(5.8)**

PSNR=23.31 dB **(5.9)**

Results of the comparison between the original image and the reconstructed image:

MSE=55 **(5.10)**

PSNR=30.72 dB **(5.11)**

5.2.1.3. When 1 packet from every flow is lost and they had neighbour info

Fig 5.5 Image with one lost packet from every flow that form a little green block and its reconstruction with the “Algorithm P”

Fig 5.6 Zooms of one part of the last two images

Results of the comparison between the original image and the image with losses:

$$MSE=92 \quad \textbf{(5.12)}$$

$$PSNR=28.49\ dB \quad \textbf{(5.13)}$$

Results of the comparison between the original image and the reconstructed image:

$$MSE=61 \quad \textbf{(5.14)}$$

$$PSNR=30.27 \quad \textbf{(5.15)}$$

5.2.1.4. When 1 burst of 25 packets from different flows are lost and they had neighbour info

Fig 5.7 Image with 25 lost packets from every flow that form a big green block and its reconstruction with the “Algorithm P”

Fig 5.8 Zooms of one part of the last two images

Results of the comparison between the original image and the image with losses:

MSE=287 **(5.16)**

PSNR=23.55 dB **(5.17)**

Results of the comparison between the original image and the reconstructed image:

MSE=264 **(5.18)**

PSNR=23,91 dB **(5.19)**

As it is seen, the PSNR results do no present big changes between the obtained from the image with losses and the obtained from the image reconstructed. This is because the PSNR does the comparisons with all pixels of the images, being few lost pixels compared with all the pixel-maps.

Furthermore, these results can create confusion because in several cases the good results of the PSNR have not coincided with good subjective results which can be obtained with the human perception.

Anyway, the reconstructions have shown a good quality and this is the followed objective. Only in the strange cases where the green blocks are formed, the module of reconstruction has been exceeded. (See chapter for the probabilities)

5.2.2. Results obtained with the “Algorithm F”

5.2.2.1. When 1 flow is cut in a transmission of 4 flows

Fig 5.9 Image received in 4 flows with one of them lost, and its reconstruction with the “Algorithm P”

Fig 5.10 Zooms of one part of the last two images

Results of the comparison between the original image and the reconstructed image:

$$MSE=66 \quad \textbf{(5.20)}$$

$$PSNR=29.93\ dB \quad \textbf{(5.21)}$$

5.2.2.2. When 1 flow is cut in a transmission of 9 flows

Fig 5.11 Image received in 9 flows with one of them lost, and its reconstruction with the “Algorithm P”

Fig 5.12 Zooms of one part of the last two images

Results of the comparison between the original image and the reconstructed image:

MSE=59 **(5.22)**

PSNR=30.42 dB **(5.23)**

5.2.2.3. When 1 flow is cut in a transmission of 16 flows

Fig 5.13 Image received in 16 flows with one of them lost, and its reconstruction with the “Algorithm P”

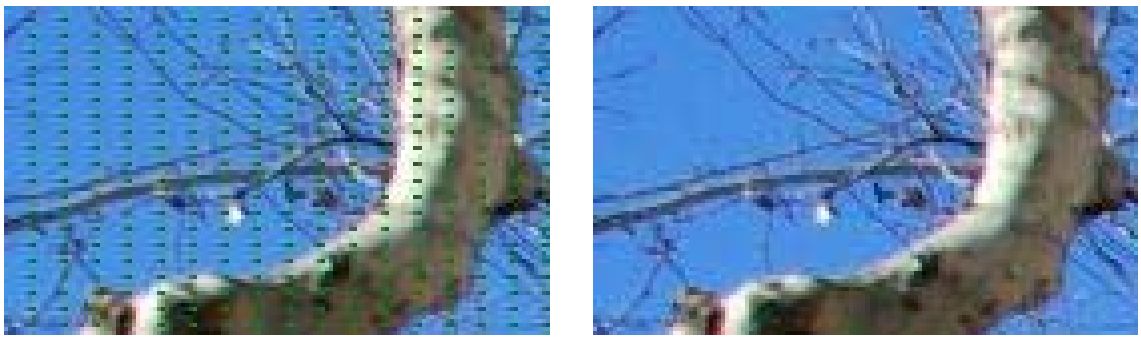


Fig 5.14 Zooms of one part of the last two images

Results of the comparison between the original image and the reconstructed image:

MSE=57 **(5.24)**

PSNR=30.57 dB **(5.25)**

5.2.2.4. When 1 flow is cut in a transmission of 25 flows

Fig 5.15 Image received in 25 flows with one of them lost, and its reconstruction with the “Algorithm P”

Fig 5.16 Zooms of one part of the last two images

Results of the comparison between the original image and the reconstructed image:

MSE=56 **(5.26)**

PSNR=30.64 dB **(5.27)**

5.2.2.5. When 1 flow is cut in a transmission of 36 flows

Fig 5.17 Image received in 36 flows with one of them lost, and its reconstruction with the “Algorithm P”

Fig 5.18 Zooms of one part of the last two images

Results of the comparison between the original image and the reconstructed image:

MSE=56 **(5.28)**

PSNR=30.64 dB **(5.29)**

5.2.2.6. When 1 flow is cut in a transmission of 64 flows

Fig 5.19 Image received in 64 flows with one of them lost, and its reconstruction with the “Algorithm P”

Fig 5.20 Zooms of one part of the last two images

Results of the comparison between the original image and the reconstructed image:

$$MSE=55 \quad \textbf{(5.30)}$$

$$PSNR=30.72\ dB \quad \textbf{(5.31)}$$

The reconstructions with the “Algorithm F” gives fine results in all cases, objective and subjectively. The PSNR is always around 30 dB and it is important because the volume of green pixels that appears when one flow is cutted is important. The point 5.2.2.1 needs a special mention because shows the worst case where one flow is cutted from one transmission made with 4. The 25% of the information is lost and is created with the algorithm getting good visual and numerical results.

CONCLUSIONS AND PERSONAL EVALUATIONS

Once finished the project, some conclusions and personal evaluations from the work of the last months can be extracted:

Starting to work in one project when it is in developed since some years ago is always a challenge. To be able to create one module of this, the knowledge of its internal functions is needed and it always is a long long process.

The module implemented is the first approach in the V3 project of i2cat on solving problems against losses and this module is not ready to be included in a stable version of Ultragrid. However this project marks the first step on the problem of losses that can appear in this platform.

In the telecommunications environments the losses always can appears in the transmissions but sometimes these losses can be predicted and avoided as is done with the use of “Algorithm F” or in a worst case these can be minimized or completely deleted as is done with the “Algorithm P”.

In other hand, the importance of the probabilities can be seen in the project. All cases have to be contemplated but with one study of probabilities many of them can be ignored.

Finally and for closing the conclusions and the personal valuations, one thing more can be extracted from this project, in the world of the digital reconstruction image, the interpolation techniques are the most common tools in this kind and they have been applied in this project. To validate the quality of the reconstruction the PSNR has been used, giving objective values through a mathematical operation pixel by pixel. In this project has been seen that sometimes one good result from this tool no represents a good visualization of the video. In the future, the study of the image reconstructions from the human perception point would have to be something priority.

REFERENCES

[1] V3 project.
http://www.i2cat.cat/i2cat/servlet/I2CAT.MainServlet?seccio=21_61

[2] i2cat foundation.
http://www.i2cat.cat/i2cat/servlet/I2CAT.MainServlet

[3] HDwiki. SAGE.
http://hdwiki.i2cat.net/mediawiki/index.php/SAGE

[4] F. J. Iglesias Garcia, “Development of an integrated interface between SAGE and UltraGrid”.
http://upcommons.upc.edu/pfc/bitstream/2099.1/6068/1/memoria.pdf

[5] D. Turull Torrents, “Performance and enhancement for HD videoconference environment”.
http://upcommons.upc.edu/pfc/bitstream/2099.1/5000/1/memoria.pdf

[6] Wikipedia. High definition Television.
http://en.wikipedia.org/wiki/High-definition_television

[7] Wikipedia. Modelo de color RGB.
http://es.wikipedia.org/wiki/Modelo_de_color_RGB

[8] Wikipedia. RGBA color space.
http://en.wikipedia.org/wiki/RGBA_color_space

[9] Wikipedia. YUV.
http://es.wikipedia.org/wiki/YUV

[10] Wikipedia. PSNR.
http://en.wikipedia.org/wiki/PSNR

[11] Wikipedia. MSE.
http://en.wikipedia.org/wiki/Mean_squared_error

[12] Wikipedia. Interpolation
http://en.wikipedia.org/wiki/Interpolation

[13] Digital image interpolation
http://www.cambridgeincolour.com/tutorials/image-interpolation.htm

[14] Pratt, “Digital image processing”, (2002)

[15] Gonzalez, “Tratamiento digital de imagenes”, (1996)

[16] Marco A. Peña Basurto, Jose M. Cela Espin, “Introducción a la programación en C”, Ed. UPC, Barcelona, (2000)

ANNEX A: SUMMARY OF THE CODE

Colour legend

IMPORTANT DECLARATIONS
DETECTION OF LOSSES
ALGORITHM P (PAACKETS)
ALGORITHM F (FLOWS)

FUNCTIONS
COMMENTS

```
#include "config.h"
#include "config_unix.h"
#include "config_win32.h"
#include "debug.h"
#include "rtp/rtp.h"
#include "rtp/rtp_callback.h"
#include "rtp/pbuf.h"
#include "rtp/decoders.h"
#include <host.h>

#include "video_types.h"

#define SIZE_OF_48K_AUDIO     (48*1024)
#define BUFFER_SIZE_INDEX     (SIZE_OF_48K_AUDIO-4)
#define CHANNEL_START_INDEX   (SIZE_OF_48K_AUDIO-12)
#define BYTES_PER_SAMPLE      (6*4) // 24 Bytes = 6 channel * 32bits(=
24bits[=pure sample] + 7bits[=zeros])
extern video_frame_rate FrameRate;
int frame_num=0;


/*****************************************************************/

#define NUMF 64

int current_pckt[NUMF];
int last_pckt[NUMF];

int pos[NUMF];

struct
{
      int offset[1000][NUMF];
      int len[1000][NUMF];
      int num[NUMF];
      int total_num;
}lost_pckts[3];

int primer_frame=1;
int pasado=0;
int actual=1;
int arreglo=0;
```

```
/*****************************************************************/

static void copy_video_p2f (unsigned char *frame, rtp_packet *pckt){


uint32_t HD_BYTES=16;
uint32_t HD_PIXELS=6;

char *offset;
payload_hdr_t *curr_hdr;
payload_hdr_t *hdr[20];
int hdr_count = 0, i,j;
int frame_offset = 0;
char *base;
int len;
int flow=0;
int f1,f2;
int x_flow;
int y_flow;

if (FrameRate == VIDEO_FRAME_8BIT) {
      HD_BYTES  = 4;
      HD_PIXELS = 2;
}
//DATA EXTRACTION FROM PACKETS

/* figure out how many headers ? */
curr_hdr = (payload_hdr_t *) pckt->data;
while (1) {
      hdr[hdr_count] = curr_hdr;
//    printf("x: %x \n", ntohs (curr_hdr->x_offset));
//    printf("y: %x \n", ntohs (curr_hdr->y_offset));
//    printf("l: %x \n", ntohs (curr_hdr->length));
//    printf("hdr_count: %d \n", hdr_count);
//    printf("comprueba:   %X   \n",   (ntohs(curr_hdr->x_offset)   &
(1<<15)));
//    printf("comprueba_sin: %X \n", (ntohs(curr_hdr->x_offset)));
      hdr_count++;
      if ((ntohs(curr_hdr->flags) & (1<<15)) != 0) {
                  /* Last header... */
                  break;
      }
      if (hdr_count == 20) {
                  /* Out of space... */
                  break;
      }
      curr_hdr++;
}


offset=(char *) (pckt->data) + hdr_count * 8;


/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
```

```
f2=(ntohs(hdr[0]->x_offset)&0x7000)>>12;
f1=(ntohs(hdr[0]->y_offset)&0x7000)>>9;
flow= f1|f2;
int final;
final=(dp_map->columns*2*dp_map->lines)/(dp_map->num*dp_map->num);


current_pckt[flow]=((ntohs(hdr[0]->x_offset)&0x0FFF)*2+(ntohs(hdr[0]-
>y_offset)&0x0FFF)*(dp_map->columns*2/dp_map->num));

int total_len=0;
for(i=0;i<hdr_count;i++){
      total_len=total_len+ntohs((hdr[i]->length)) & 0x3FFF;
}


if((last_pckt[flow]==(final))                                          &&
(current_pckt[flow]+total_len)!=(final)){

      lost_pckts[actual].offset[pos[flow]][flow]=current_pckt[flow]+to
      tal_len;

      lost_pckts[actual].len[pos[flow]][flow]=last_pckt[flow]-
      (current_pckt[flow]+total_len);
      lost_pckts[actual].num[flow]++;
      lost_pckts[actual].total_num++;
      pos[flow]++;
      printf("1-Falta    el    paquete    de    la    posicion    %d\n",
      lost_pckts[actual].offset[pos[flow]-1][flow]);

}

else if((last_pckt[flow]!=(final))){
      if(current_pckt[flow]+total_len!=last_pckt[flow]){
            lost_pckts[actual].offset[pos[flow]][flow]=current_pckt[flo
            w]+total_len;//+1;
            lost_pckts[actual].len[pos[flow]][flow]=last_pckt[flow]-
            (current_pckt[flow]+total_len);
            lost_pckts[actual].num[flow]++;
            lost_pckts[actual].total_num++;
            pos[flow]++;
            printf("2-Lost_Packet[%d][%d]=%d\t,  Lost  lenght  info:  %d,
            rafagas:                  %d\n",                  (pos[flow]-
            1),flow,lost_pckts[actual].offset[pos[flow]-
            1][flow],lost_pckts[actual].len[pos[flow]-1][flow],
            lost_pckts[actual].total_num);
      }
}

//printf("Numero de sequencia: %d ---- Flujo: %d ---- Current_offset:
%d ---- Last_offset: %d ---- Info_lenght: %d ----  Pos: %d\n", pckt-
>seq, flow, current_pckt[flow],last_pckt[flow],total_len, pos[flow]);

last_pckt[flow]=current_pckt[flow];

/*****************************************************************/

//DATA SAVED IN THE CORRECT POSITION TO BE SENT TO THE SCREEN

x_flow=flow%dp_map->parts;
```

```
y_flow=flow/dp_map->parts;

for(i=0;i<hdr_count;i++){

      len=ntohs((hdr[i]->length))&0x3FFF;

      frame_offset   =(int)(   frame   +      x_flow*4+(ntohs(hdr[i]-
>x_offset)&0x0FFF)*dp_map->num*2+((ntohs(hdr[i]->y_offset)&
0x0FFF)*dp_map->num+y_flow)*dp_map->columns *2);
      for(j=0;j<len;j=j+4){

      base=frame_offset+j*dp_map->num;
      memcpy(base,offset+j,4);
      }
offset+=len;

}

}


void decode_frame(struct coded_data *cdata, unsigned char *frame, char
*pt_data_new, int mediaFlag){


int i,j,k;
int final;
int imagen_perdida[NUMF];
int num_imagenesp;//0

final=(dp_map->columns*2*dp_map->lines)/(dp_map->num*dp_map->num);
//Offset del ultimo pixel de la subimagen

for(i=0;i<(dp_map->num)*(dp_map->num);i++){
      last_pckt[i]=(final);
      pos[i]=0;
      lost_pckts[actual].num[i]=0;
}
lost_pckts[actual].total_num=0;


/*****************************************************************/

//BUCLE OF COPY PACKET DATA TO THE SDL BUFFER

while (cdata != NULL) {
      if(mediaFlag==1){
            if(decode==TRUE) copy_video_p2f(pt_data_new2, cdata->data);
            copy_video_p2f(frame, cdata->data);
      }

      else {
            //audio
            copy_audio_p2f(frame, cdata->data);
      }
      cdata = cdata->nxt;
}

/*****************************************************************/
```

```
//SOME INITIALITZATIONS AND CONTROL OF LOST FLOWS

num_imagenesp=0;
for(i=0;i<(dp_map->num)*(dp_map->num);i++){
      imagen_perdida[i]=0;
      if(last_pckt[i]==(final)){
            imagen_perdida[i]=1;
            num_imagenesp++;
            printf("Perdida la subimagen %d entera.\n", i);
      }
}

//CONTROL OF LOST PACKETS IN THE DECODER FUNCTION

if(lost_pckts[actual].total_num>0){
      if(primer_frame==1){
            lost_pckts[2]=lost_pckts[actual];
            arreglo=1;
            //printf("InterpolaciÃ³n necesaria, hay perdidas en el
            frame 0:\n");
      }
      else{
            int inicio;
            lost_pckts[2].total_num=0;
            for(i=0;i<(dp_map->num)*(dp_map->num);i++){
                  inicio=0;
                  lost_pckts[2].num[i]=0;
                  for(j=0;j<lost_pckts[actual].num[i];j++){
                        for(k=0; k<lost_pckts[pasado].num[i];k++){

                              if(lost_pckts[actual].offset[j][i]==lost_
                              pckts[pasado].offset[k][i]){
                              lost_pckts[2].offset[inicio][i]=lost_pckt
                              s[actual].offset[j][i];
                              lost_pckts[2].len[inicio][i]=lost_pckts[a
                              ctual].len[j][i];
                              inicio++;
                              arreglo=1;
                              //printf("InterpolaciÃ³n necesaria, las
                              perdidas actuales y anteriores
                              coinciden:\n");
                              }
                        }
                  }
                  lost_pckts[2].num[i]=inicio;
                  lost_pckts[2].total_num+=inicio;
            }
      }

if(arreglo==1){

int frame_offset;
char *base;
int x_offset;
int y_offset;
int x_flow;
int y_flow;
int salto_linea=0;
unsigned char uu,ud,ul,ur,vu,vd,vl,vr,yl2,yr1,yu1,yu2,yd1,yd2;
```

```
float umedia,vmedia,y1media,y2media;
int j=0;
int x=0;

//printf("Total paquetes perdidos a interpolar: %d\n",
lost_pckts[2].total_num);

for(i=0;i<(dp_map->num)*(dp_map->num);i++){
	x_flow=i%dp_map->parts;
	y_flow=i/dp_map->parts;
	printf("xflow: %d\n", x_flow);
	printf("Del flujo[%d]: %d\n", i, lost_pckts[2].num[i]);
	for(j=0;j<lost_pckts[2].num[i];j++){

		y_offset=lost_pckts[2].offset[j][i]/(dp_map-
		>columns*2/dp_map->num);

		x_offset=lost_pckts[2].offset[j][i]%(dp_map-
		>columns*2/dp_map->num);

		frame_offset=(x_flow*4+(x_offset)*dp_map-
		>num+(y_offset*dp_map->num+y_flow)*dp_map->columns*2);

		//printf("offset grande: %d\n", frame_offset);

		for(x=0;x<lost_pckts[2].len[j][i];x=x+4){
			//printf("len[%d][%d]:%d\n", i, j,
			lost_pckts[2].len[i][j]);

			if((frame_offset%(1920*2)+x*dp_map->num-
			x_flow*4)%(1920*2)==0){
				salto_linea=salto_linea+dp_map->num-1;
				//printf("salto: %d\n",salto_linea);
			}

			base=(int)frame+frame_offset+dp_map-
			>num*x+salto_linea*1920*2;


			//printf("base: %p\n",base);

			uu=*(base-(1920*2));
			ud=*(base+(1920*2));
			ul=*(base-4);
			ur=*(base+4);
			vu=*(base-(1920*2)+2);
			vd=*(base+(1920*2)+2);
			vl=*(base-2);
			vr=*(base+6);
			yl2=*(base-1);
			yr1=*(base+5);
			yu1=*(base-(1920*2)+1);
			yu2=*(base-(1920*2)+3);
			yd1=*(base+(1920*2)+1);
			yd2=*(base+(1920*2)+3);
			umedia=((float)ul+(float)ur+(float)uu+(float)ud)/4;
			vmedia=((float)vl+(float)vr+(float)vu+(float)vd)/4;
			y1media=((float)yl2+(float)yu1+(float)yd1)/3;
			y2media=((float)yr1+(float)yu2+(float)yd2)/3;
			*(base)=(unsigned char)umedia;
			*(base+1)=(unsigned char)y1media;
```

```
                        *(base+2)=(unsigned char)vmedia;
                        *(base+3)=(unsigned char)y2media;
                }
                salto_linea=0;
        }
}

}
arreglo=0;
printf("\n");
}

//SWAP OF THE LOST_PACKETS VECTOR POSITIONS

if(actual==1){
        actual=0;
        pasado=1;
}
else{
        actual=1;
        pasado=0;
}

primer_frame=0;


/********************* Reconstrucción por flujos ********************/

if((num_imagenesp>0)){

printf("Num imag p:%d\n", num_imagenesp);
int frame_offset=0;
char *base;
int i,j;
int a, b;
unsigned char uu,ud,ul,ur;
unsigned char vu,vd,vl,vr;
unsigned char yl1,yl2,yr1,yr2,yu1,yu2,yd1,yd2;
float umedia,vmedia,y1media,y2media;

/*************************1 flujo perdido ************************/

if(num_imagenesp==1){

/*Offset inicial*/
/*a columna, b fila*/

/*********************************************************************/

        if(dp_map->num==2){
                a=4;
                b=1;
        }

        if(dp_map->num==3){
                a=8;
                b=2;
        }

        if(dp_map->num==4){
                a=12;
```

```
                b=3;
        }

        if(dp_map->num==5){
                a=16;
                b=4;
        }

        if(dp_map->num==6){
                a=20;
                b=5;
        }

        if(dp_map->num==7){
                a=24;
                b=6;
        }

        if(dp_map->num==8){
                a=28;
                b=7;
        }

/*****************************************************************/

for(j=b;j<(1080);j=j+b+1){
      for(i=a;i<(1920*2);i=i+a+4){
                frame_offset                    =                   (j*1920*2+i);
                base=frame+frame_offset;
                uu=*(base-(1920*2));
                ud=*(base+(1920*2));
                ul=*(base-4);
                ur=*(base+4);
                vu=*(base-(1920*2)+2);
                vd=*(base+(1920*2)+2);
                vl=*(base-2);
                vr=*(base+6);
                yl2=*(base-1);
                yr1=*(base+5);
                yu1=*(base-(1920*2)+1);
                yu2=*(base-(1920*2)+3);
                yd1=*(base+(1920*2)+1);
                yd2=*(base+(1920*2)+3);

                umedia=((float)ul+(float)ur+(float)uu+(float)ud)/4;
                vmedia=((float)vl+(float)vr+(float)vu+(float)vd)/4;
                y1media=((float)yl2+(float)yu1+(float)yd1)/3;
                y2media=((float)yr1+(float)yu2+(float)yd2)/3;
                *(base)=(unsigned char)umedia;
                *(base+1)=(unsigned char)y1media;
                *(base+2)=(unsigned char)vmedia;
                *(base+3)=(unsigned char)y2media;
                }
        }
}

}
}
num_imagenesp=0;

}
```