POLITECNICO
DI TORINO


MASTER THESIS

# Video Streaming Optimization For The A380 Cabin Video Monitoring System

*Author:*
Amna BEN AYED
*Supervisor:*
Guillaume CHAUVEL- Zodiac Aerospace
*examinators:*
Enrico MASALA - Politecnico di Torino
Yves MATHIEU - Télécom ParisTech

*Host organism:*


September 13, 2017

# *Abstract*

**Video Streaming Optimization For The A380 Cabin Video Monitoring System**

by Amna BEN AYED

Statistics claim that the aircraft video surveillance systems market is estimated to be $127.13 million in 2014 and is expected to reach $173.83 million by 2020. This increasing concern for in-flight surveillance is triggered by the growing emphasis on the security of the aircraft and the safety of passengers. The need for video surveillance is even greater for very large aircrafts like the A380 which is the world's largest passenger airliner. The system is usually implemented within the most strategic areas of the aircraft, including the passenger cabins, the cockpit door and cargo areas. In order to design a video surveillance system that is efficient and that is also suitable for safe flight, various challenges are encountered.

This work focuses on one challenge of key importance in the performance and quality of the whole system, which is the streaming of videos from surveillance cameras to aircraft monitors. This work was done during my 6 month internship at Zodiac Aerospace, a world leader in aerospace equipments. It is mainly intended to provide a quite simple and practical solution to improve and optimize the video streaming transport layer currently implemented in the Zodiac A380 video surveillance system. The proposed optimization makes use of a common practice in the multimedia world which is putting multimedia content in a convenient form allowing it to be easily retrieved and correctly played. In fact, in order to get the best streaming performance, it is absolutely necessary to find the best way to compress and encode multimedia streams. But this is not sufficient. Transmitting the streams in an adequate multimedia container plays on its side a key role in determining the quality of videos being streamed. This was realized by taking advantage of the characteristics of a standard multimedia container format which has been selected and adapted for the specific case of the aircraft cabin video monitoring system.

# *Acknowledgements*

I would first like to thank my supervisor in Zodiac Mr Guillaume Chauvel and all the CVMS team for supporting me upon the different stages of my internship.

I would also like to thank Professor Enrico Masala from Politecnico di Torino and Professor Yves Mathieu from Télécom ParisTech for participating in the validation of this work.

Finally, I would like to extend my deepest gratitude to my family and friends for all the love they gave me and for their continuous encouragement throughout my years of study. Without them I wouldn't be where I am today. Thank you...

*Amna Ben Ayed*

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **ADU** | **A**cquisition **D**ata **U**nit |
| **CVMS** | **C**abin **V**ideo **M**onitoring **S**ysytem |
| **CDSS** | **C**ockpit **D**oor **S**urveillance **S**ystem |
| **CDAM** | **C**entralized **D**ata **S**urveillance **S**ystem |
| **CMV** | **C**oncentrator and **M**ultiplexer for **V**ideo |
| **DVR** | **D**igital **V**ideo **R**ecorder |
| **DAL** | **D**evelopment **A**ssurance **L**evel |
| **DAU** | **D**ata **A**cquisition **U**nit |
| **FAP** | **F**light **A**ttendant **P**anels |
| **GSP** | **G**round **S**ervice **P**anel |
| **HTTP** | **H**yper **T**ext **T**ransfer**P**rotocol |
| **HD** | **H**igh **D**efinition |
| **LRU** | **L**ine **R**eplaceable **U**nit |
| **LMF** | **L**ocal **M**aintenance **F**unction |
| **MSOC** | **M**aster **S**ystem **O**n **C**hip |
| **MJPEG** | **M**otion **JPEG** |
| **SATCOM** | **Sat**ellite **Com**munication |
| **SD** | **S**tandard **D**efinition |

# Chapter 1

# Introduction

## 1.1 Thesis Context

My master thesis has taken place in Zodiac Aerospace Group. I was part of the software development team participating in the realization of the Zodiac video surveillance system for the airbus aircraft A380. In this chapter I will present the company and the project as well as the organization and the plan of my thesis.

## 1.2 Company Presentation

Zodiac Aerospace is a leader in aerospace equipments. It is specialized in realizing on-board systems for aircrafts as well as helicopters. The company was founded by Maurice Mallet in 1896. Now it employs around 35,000 people in 100 different sites around the world. Zodiac Aerospace has four main entities: Zodiac Aerosystems, Zodiac Cabin Zodiac Seats, Zodiac Aerospace Services. My internship was with Zodiac Aerotechnics which is a part of the entity Zodiac Aerosystems. Zodiac Aerotechnics - Sensing and System Management (ZSSM) is a designs and manufactures highly critical sensors and monitoring systems for the both aircrafts and helicopters. The entity also builds complex aerospace systems integrating fuel gauging, automatic refueling control, center of gravity management and cockpit display functions.

## 1.3 Cabin Video Monitoring System Project

Getting the aircraft equipped with a video surveillance system plays a key role in the safety and security of passengers. The importance of video surveillance is even greater in an aircraft with a very large capacity like the A380.

FIGURE 1.1: Entirely supervised cabin using a camera with a wide angle of view

The system's been required for 80% of A380 aircrafts and has always been supplied by Parker who has recently decided to stop its production line of the system. Hence in a response to Airbus request for proposal, Zodiac Controls, part of Zodiac Group, got engaged in supplying the equipment which perfectly comes as a complement to the group's offer in terms of control systems.

The system development budget is around 7.14M$. A part's been paid by Airbus (around 3.6M$) and the other part is expected to be paid by selling the CVMS Zodiac HD system and by replacing the Parker equipments.

The Zodiac system implements the High Dynamic Range imaging technology which makes a great improvement with respect to the Parker system which only implements Simple Definition video streaming.


The Zodiac system also guarantees compatibility with the previous one. This is a key point as nearly 80 aircrafts are already equipped with the Parker system and in case an equipment fails it is necessary to be able to easily replace it without further modifications. This is the wide context of my internship. During my internship I have mainly worked on the SD video streaming part.

## 1.4 Project Team Presentation

The production of a certified aeronautical equipment requires a multi-functional project team comprised of all development engineering disciplines, Manufacturing Engineering, Quality Assurance (hardware/software), Material (procurement/ product planning), Manufacturing Support, Contracts, and Administration under the Project direction Lead. The main roles and functions delegated to the members of the A380 CVMS project team are:

- Project Manager:
  in cooperation with the Software manager , the System Manager and the Software product Team Manager defines and set up the necessary and adequate facilities for implementation of the life cycle processes. He plan the attribution of resources in cooperation with the Department management and checks that the target delivery dates can be observed.

- System manager (SYM):
  defines the equipment requirements, allocates them to the software or to the hardware.

- Software product team manager (SWPTM):
  ensures the physical and functional completeness of the configuration components in the reference space. He performs configuration status accounting activities in compliance with the approved plans and procedures in coordination with the SWM and draws up the baselines.

- Software manager (SWM):
  performs the estimations of the development effort, establishes and updates the plans and the schedule of the activities, coordinates the work of the software development and validation and verification teams and participates in the project reviews.

- Software development team:develops the source code for the software product.

- Software validation & verification team:
  develops the software high-level Test Cases, develops the software high-level Test Procedures, validates the software high-level Test Cases, validates the software high-level Test Procedures, executes the Test Procedure and records the Test Results, verifies Test Results, reports the validation and verification results.

- Software configuration manager:
  ensures the physical and functional completeness of the configuration components in the reference space. He performs configuration status accounting activities in compliance with the approved plans and procedures in coordination with the SWM and draws up the baselines.

- Software methodology manager (SWMM ):
  Provides to the SWPTM the tools, techniques and methods appropriate to the project. He provides the support necessary to all those involved for the application of the methodological documents and the use of the reference and tool software library.

- System quality assurance manager (SWQAM ):
  ensures the conformity with the contractual requirements of the applicable plans and procedures in cooperation with the SWM, the SWPTM, the SWMM and the customer. He evaluates their application all through the life cycle in compliance with the PSAC. The SWQAM has the authority to guarantee independently that the corrective actions are performed. He is directly accountable only to the Quality management.
- Hardware team manager:
  performs the estimations of the development effort, establishes and updates the plans and the schedule of the activities, coordinates the work of the software development and validation and verification teams and participates in the project reviews.
- Sales manager: builds the sales plan.
- Computer system manager: manages the general computer equipment environment of the project and provides all those involved with the support necessary for their configuration and use.

During my internship, I was part of the software development team but I was also in close interaction with the hardware team mainly the Zodiac camera architect.

## 1.5 Thesis Organization

In order to achieve the goals of my internship I went through the following steps:

- Getting a full understanding of the system in particular the different software components.
- Getting used to manipulating the test bench of the system which provides simulators for the different system components and an environment quite close to the one on-board an aircraft.
- Analyzing the system behavior and identifying all issues related to the CVMS's video streaming function.
- Proposing solutions to optimize video streaming and visualization.
- Solution implementation.
- Analyzing the performance of the system after integrating the new solution and comparing it with respect to the original version.

# Chapter 2

# Cabin Video Monitoring System Overview

## 2.1 System Development Methodology

The life cycle of the overall CVMS system follows the V-Cycle model as described in the figure below. As I have mainly intervened on the software part of the CVMS I will focus here on the methodology used for software development.

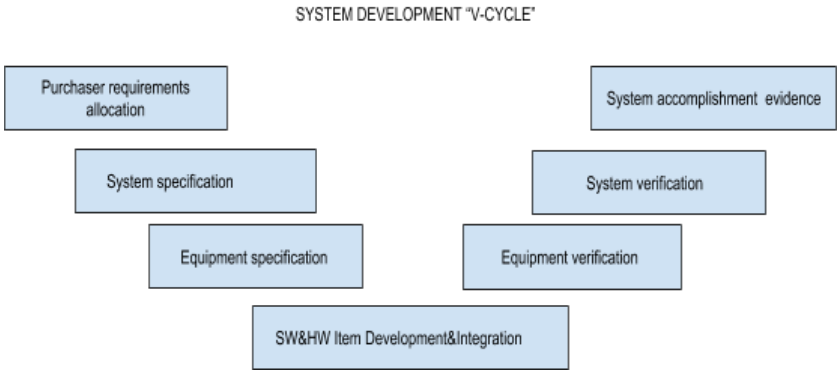



FIGURE 2.1: System development V-Cycle.

### 2.1.1 Software life cycle

Different processes are needed in order to assure the quality of a certified software product. the CVMS department summarizes them as follows:

- **1.** Equipment definition (provides data for SW development),Planning.
- **2.**Development and Validation. (specification, design, coding and integration/ debugging)
- **3.**Verification (development and validation of test procedures, tests, verification).
- **4.**Configuration management.
- **5.**Coordination for the certification.
- **6.**Project management.
- **7.**Quality assurance.

### 2.1.2 Product delivery baselines

All along the software development, prototypes are realized and tested by Airbus before the final product can be released. Testing on prototypes is necessary in order to detect and correct potential defects on the software.

**Ground Test Prototype:**Once the equipment validation is completed, a software prototype is delivered to Airbus for ground testing. Various documents should be delivered along with the software source code and executable files. The documents include, among others, the Software Design document, the Software Validation and Verification document, the software Configuration index, etc.

**First Flight Prototype:** This baseline is accomplished after successful completion of the equipment validation. The software is tested and is still subject to changes. As for Ground Test prototype, a number of documents with the source code and executable files are delivered.

**Product release:** the developed software end item is released and accepted by Airbus. This baseline shall contain the complete software documentation.

### 2.1.3 Certification considerations

We should also note that since it consists of an avionic system, it is necessary to go through the certification process which assures that all the airworthiness requirements are respected. These requirements are based on the Development Assurance Level (DAL) assigned to the equipment from the safety assessment process and hazard analysis. A DAL D has been assigned to all CVMS software components. The certification process's been based on the standard guideline DO-178B, Software Considerations in Airborne Systems and Equipment Certification and it is consists mainly:

- a.On the establishment of software plans at the beginning of the life cycle, in compliance with the requirements of document ED-12B/DO-178B.
- b.On the demonstration of the implementation of these software plans in the course of the project.

## 2.2 Software Development Tools

### 2.2.1 Project Management Systems

- **Git:**It is a mature and widely used modern version control system. Each developer may clone the remote git repository it is interested in on his local machine, making regular commits allows him to save the project history and browse it when needed to view changes between different commits. This makes going back and canceling modifications very easy and efficient. Git also allows branching which is very practical in order to test different ideas and to easily switch from one to another. Besides, Git allows to guarantee data integrity by computing a checksum for each commit.
- **Repo:** The software developed for the CVMS system is not stored as one Git repository, it is instead a number of git repositories one included in another.

Hence, Repo is mainly used to make it easy to manage many Git repositories. For example it is possible to update or clone all Git repositories of the project by only running one single command repo sync.

- **Gerrit:** It is a web-based code review tool built on top of Git. It basically allows to review commits before integrating them to the codebase. Different developers may regularly send their changes, these changes will then be pended until they are reviewed, discussed and approved by other developers.
- **Jenkins:** It is installed on the server of the shared repository where built should be run. Its role is to pick the changed source code, trigger a built and run tests if required. It is possible to browse the build output in the Jenkins dashboards. The practice of triggering a build after each commit is very common in software development and it allows detect as soon as possible build issues.

### 2.2.2 Code Development Tools

- **GNU Linux OS and software collection:** Used to implement all CVMS software components, i.e DAU and Camera softwares.
- **Development toolchain:** GNU toolchain which includes: GNU make for the automation of compilation and build, GNU Compiler Collection GCC, GNU Binutils for linking and assembling, etc.
- **Firmware build system:** PTXdist is used in order to assemble and generate firmware images. it provides a configuration system Kconfig that allows to customize package configuration.
- **Programming languages:** The major part of CVMS software is written in :
  - *C/C++*. This choice is based on the usual constraints when dealing with embedded systems which are basically: limited resources (memory and CPU), efficient memory management, small stuff to load into the system to get the software run, simple access to low level hardware API's which are generally written in C or C++.
  - *Python* is used to develop the web application based on the MVC framework Django.
  - *Java* is used to write the Java applet launched by the web application for video visualization.
  - *HTML* is used to write web pages.
- **Database server:** Redis is used as an in memory data structure to store informations needed for system functioning. It is the interface that allows the communication between different software components.

### 2.2.3 Debugging Tools

- **Mantis:** It is an open source web-based bug tracker software used to track software defects.
- **GNU Debugger GDB:** used to debug code. gdbserver is used to connect the program to be debugged to GDB.

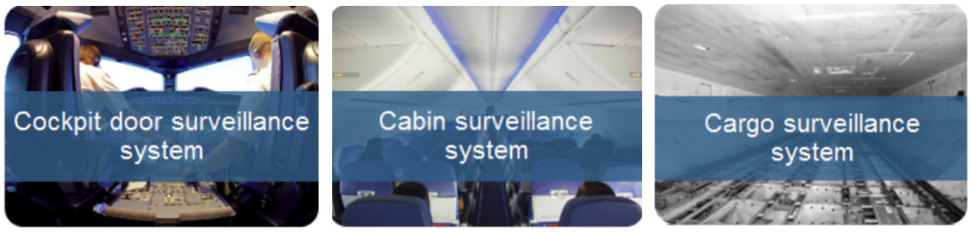
- **Valgrind:** It provides debugging and profiling tools which allow for faster and more correct programs. The most used tool is Memcheck, it allows to optimize memory management and to detect potential memory-related errors that are very common especially when coding in C and C++. This tool is very useful in avoiding errors that might lead to unpredictable behaviour and system crashes.

### 2.2.4 Network tools

- **Wireshark:** It is a free and open source packet analyzer that is used for network troubleshooting and analysis. It is a very useful tool to verify the packets exchanged between different system components.
- **PuTTy:** a free and open source terminal emulator that supports different network protocols. It was used to connect and send commands via SSH to the system embedded cards.
- **SSH Unix tool :** It is a tool for network connectivity that can be used to execute commands on a remote machine.
- **PackETH:** It is a GUI tool that allows to generate any possible Ethernet packets and to send them on an Ethernet link.

## 2.3 System general description

The CVMS is intended to assist the flight and cabin crew in order to observe all aircraft areas and get aware of any exceptional events or potential threats. For that a number of cameras are installed inside the aircraft cabins and the cockpit.Cameras can also be optionally installed in the E-Bay and cargo areas.




A crew member can watch different cabin parts at the same location by selecting the required camera stream on the Flight Attendant Panel (FAP). The cockpit crew use instead the cockpit System Display (SD) to select and watch Camera streams.

The CVMS can also be used to surveil aircraft cabin on ground. It is possible to access video streams via the Ground Service Panel (GSP). The GSP is also used to implement maintenance functions.

The CVMS is realized by an Ethernet network based on a two layer star topology connecting web cameras to the DAU passing by the ADU's network switches. The CVMS system is composed of 4 types of units :

- *Cabin Camera:* up to 30 they are installed in the entire cabin.
- *Cargo Camera:* up to 6 they are optional and should be installed in the E-Bay or in cargo areas.

- *ADU:* Area Distribution Unit. Up to 7 in the whole cabin, one ADU can be connected to up to 10 Cameras.
- *DAU:*Data Acquisition Unit. It acts as the video server of the CVMS.

Video streams are captured by CVMS cameras installed in the aircraft cockpit and cabins, they are then distributed to the system's video sinks. The figure below shows the CVMS functions involved from the acquisition of the image to its reception by the video sink.

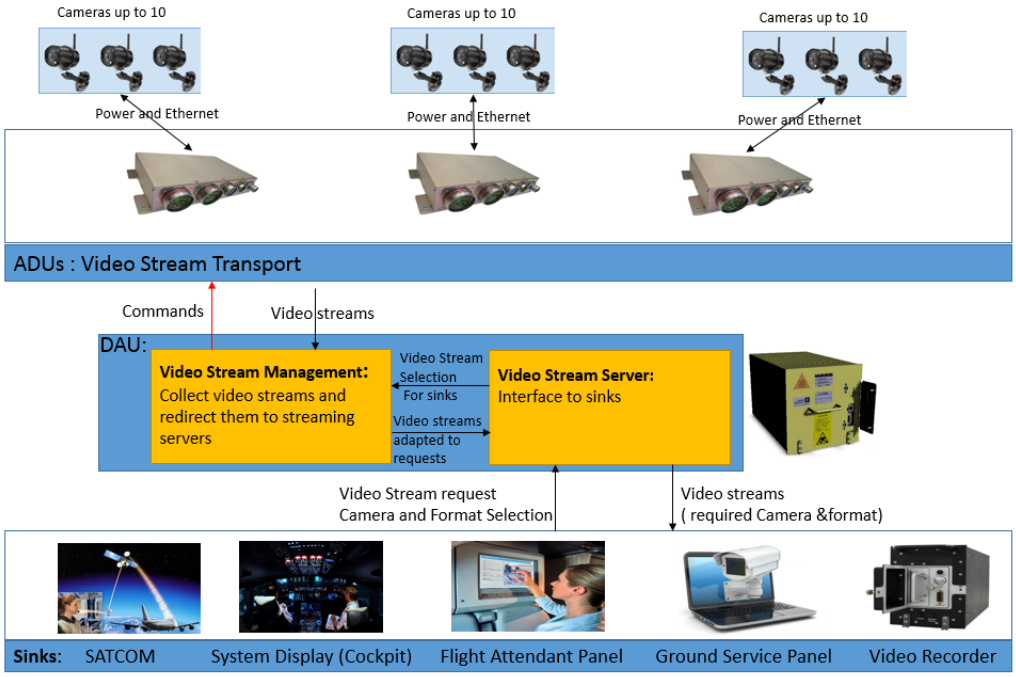



FIGURE 2.2: CVMS functional overview

The IP addresses are allocated based on the following network configuration:

```
Basis: 172.22.250
Netmask: 255.255.248.0 (/21)
Broadcast: 172.22.255.255
Range: 172.22.248.1 to 172.22.255.254
Gateway: 172.22.251.1
```

A DHCP server takes care of automatically affecting the IP address to each CVMS LRU.

## 2.4 System Basic Components

### 2.4.1 Data Acquisition Unit (DAU)

#### 2.4.1.1 Data Acquisition Unit Overview

The DAU is the core of the CVMS and it is directly or indirectly connected to all CVMS equipments and to the avionic equipments (DVR, CDAM, CMV, etc.).

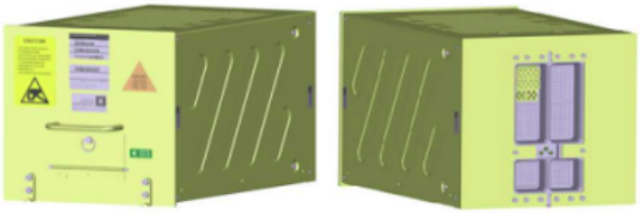


FIGURE 2.3: ARINC 600 housing the DAU

From hardware point of view the DAU consists basically of:

- several MSOCs (Media System On Chip) for all its software DAL E functions: System Control Module, FAPs/SATCOM/GSP web servers, FAPs/ CMV/Camera controllers, analog acquisition, etc.
- A microcontroller chip for its software DAL D functions: System I/O Controller.

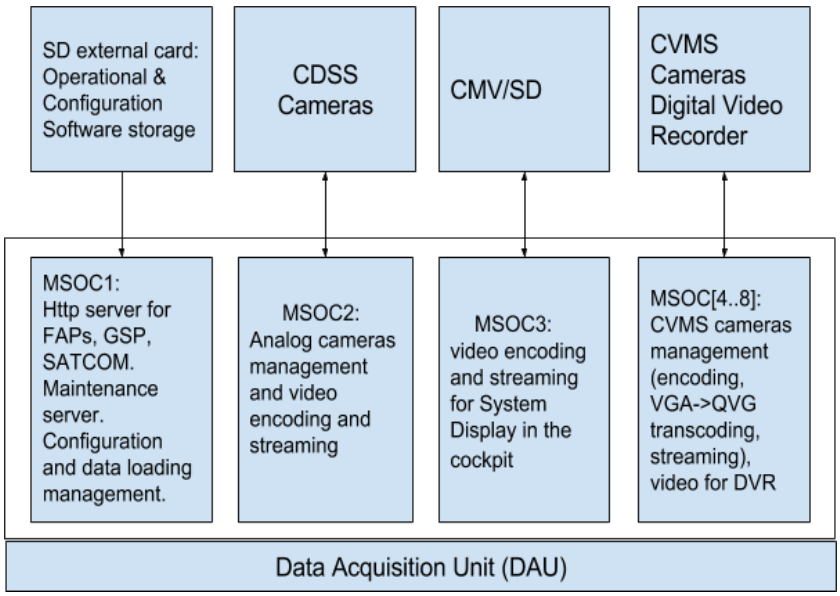



FIGURE 2.4: Data Acquisition Unit

The software component of the DAU has the following functions:

- *Video streaming:* Transmitting video streams from requested CDSS and CVMS cameras to the video sinks.
- Web application providing user interface for cabin crew and maintenance personnel on the FAPs.
- Software and configuration loading.
- Collection and display of informations on the status of different CVMS components .

Each MSOC is loaded with the system's operational software. The software image that is loaded is the same for all of them. The difference is in the processes that are activated for each one.

#### 2.4.1.2 Software loading and system Configuration

The CVMS operational software and configuration data are stored in a removable media storage. As in the figure above, an SD card is used.

- **Startup And Configuration Loading**

  At startup, the DAU checks if the stored configuration in its master SOC is similar to the default configuration in the SD external card. If they aren't similar the SD card's default configuration is copied in the DAU's internal configuration which is on its side copied to the SD current configuration. Otherwise, if they are similar, the DAU's stored configuration is compared to the SD card's current configuration.
  If they are similar The DAU's internal configuration is used, otherwise the SD's current configuration is copied to the DAU's internal configuration which is, on its side, copied to the SD's current configuration.
  At start-up, CVMS LRUs connect to the DAU configuration server to get their configuration.

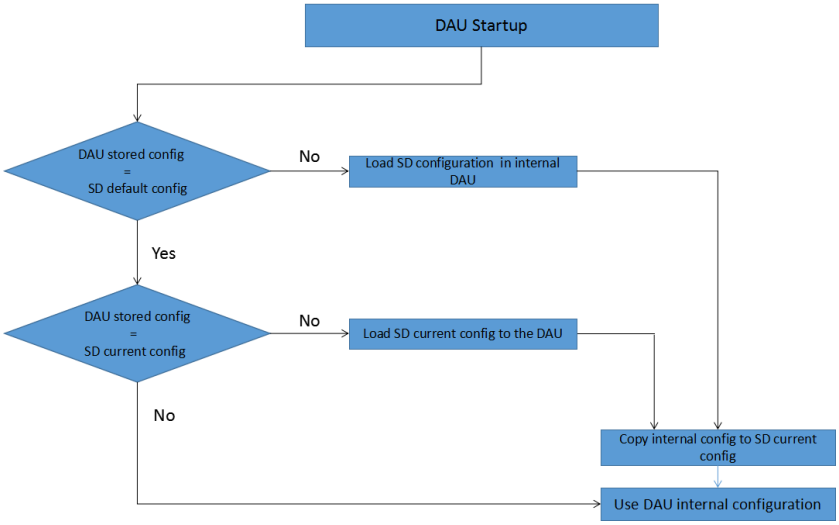



FIGURE 2.5: CVMS Configuration Loading

- **Operational Software Updating** The SD card hosts the software images for DAU, ADUs and cameras. The SD card is mounted on the DAU file system. At startup the DAU checks whether its internal software version corresponds to the one in the SD card. In case it is different, the data loading process starts.

  The DAU runs the data loading server which initiates the data loading process in all the other CVMS LRUs (Cameras and ADUs) by sending an update request to the data loading client of each of them. The update request is only sent after the verifying the software revision installed in the LRU. If it corresponds to the one in the SD card, the data loading process is not initiated in the corresponding LRU.

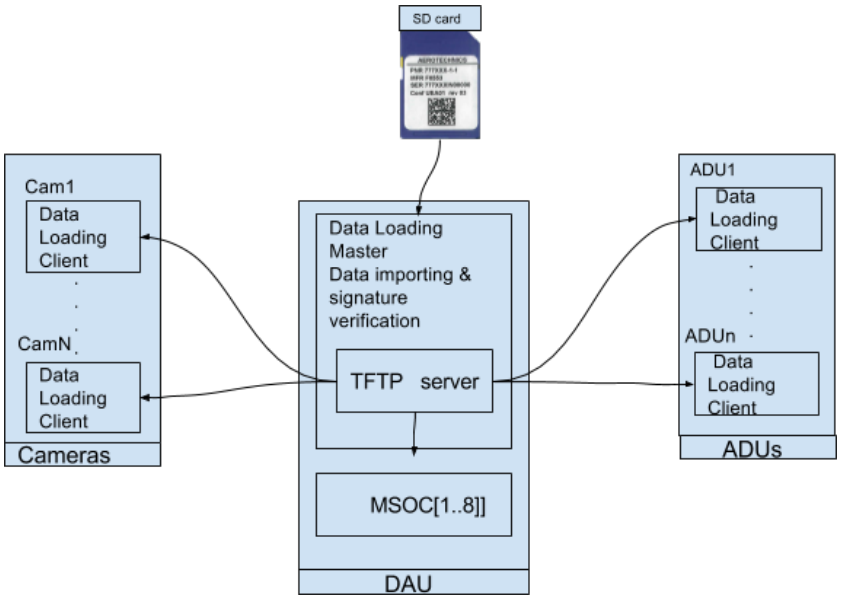



FIGURE 2.6: Operational Software Loading

#### 2.4.1.3 DAU Address allocation:

```
DAU Primary IP :   172.22.250.11
DAU Secondary SOCs : 172.22.250.[12..18]
```

### 2.4.2 Area Distribution Unit (ADU)

#### 2.4.2.1 Area Distribution Unit Overview

The ADU supplies up to 10 cameras with 28VDC and gives them an Ethernet access to the DAU. The key role of the ADUs is hence to transport video streams from Cameras to DAU. It also has other functions such as:

- Switching on or off cameras individually.
- Assigning IP addresses to its Cameras.
- Sending BITE informations to DAU.
- Participating to the software and configuration loading of CVMS.
- Data loading from DAU.

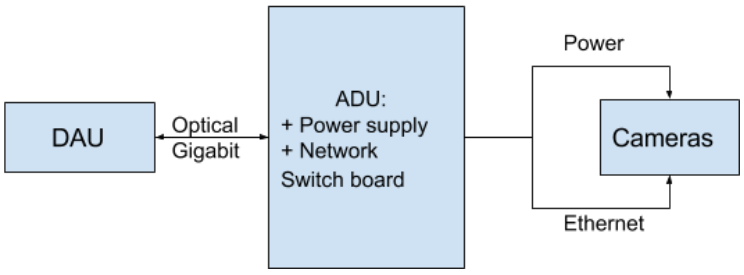



FIGURE 2.7: Area Distribution Unit

#### 2.4.2.2 ADU Address allocation

`172.22.250.[21..29]`

(address is assigned based on the DAU channel to which ADU is connected )

### 2.4.3 Cameras

#### 2.4.3.1 Camera Overview

The CVMS can be equipped with up to 30 digital cameras (cabin, cargo & custom) and 4 analog CDSS cameras. Each camera consists basically of an image sensor, an audio codec and a System on chip to run the camera software.

FIGURE 2.8: CVMS cameras

The main functions of a camera are:

- Capturing and Encoding video and audio: Depending on the video mode configured(SD, mixed or HD) the encoding algorithm used can be MJPEG or H264.
- Streaming Video and audio to the DAU: MJPEG streams are transmitted over HTTP . HD streams are transmitted over RTP.

TABLE 2.1: CVMS Video stream providers.

| Video Providers | Max number of providers |
|---|---|
| Cabin Cameras | 30 (one stream per camera) |
| Cargo Cameras | 6 (one stream per camera) |
| CDSS | 4 (analogue streams) |

#### 2.4.3.2 Camera configuration

It is possible to configure a number of parameters for each CVMS camera through the Local Maintenance Function (LMF) provided by the Ground Service Pane. These parameters are: The AEC/AGC algorithms settings, White balance settings, Image sensor settings, General image settings, Electronic steering settings, Image compression settings, Camera network settings, Camera general settings.

#### 2.4.3.3 Address allocation for cameras

Cargo cameras connected to DAU:*172.22.250.[91..100]* CVMS Cameras : *172.22.250.[101..199]* ( IP addresses are assigned based on ADU Id & Port Id to which the camera is connected )

### 2.4.4 Sinks

The CVMS is interfaced to 6 types of video sinks:

- Flight Attendant Panels (FAP) in the cabin.
- System Display (SD) in the cockpit via Concentrator and Multiplexer for Video (CMV).
- Digital Video Recorder (DVR) to record all cameras streams on a removable storage device.
- Ground Service Panel (GSP) to display the CVMS video and audio streams outside of the aircraft on ground.
- Satellite Communication (SATCOM) for transmission of video and audio streams to a ground station during flight.

FIGURE 2.9: Cockpit System Display - Flight Attendant Panel

FAPs are equipped with a touch screen to emulate the cursor. GSP can be accessed via a normal computer.

TABLE 2.2: CVMS video sinks.

| Video sink type | Max number of sinks |
|---|---|
| FAP | 4 |
| SATCOM | 1 |
| CMV | 1 |
| GDP | 1 |
| DAU Front Panel | 4 |
| SATCOM | 1 |

Video streams are formatted and converted adequately by the DAU streaming server before being sent to sinks. The table below shows the type of stream for each sink.

**Address allocation for sinks**

```
FAPs: 172.22.251.[2..11]
DVR: 172.22.250.60
SATCOM : 172.22.251.1
GSP : 172.22.250.67
```

# Chapter 3

# Video Streaming Issues

## 3.1 Video Streaming Functionality Overview

The video streams are sent from the cameras to the Area distribution units (ADU) which then forwards them to the Data Acquisition Unit (DAU). The DAU processes the video and audio streams collected from the ADU and the Cockpit Door Surveillance System (CDSS) cameras (Analog cameras directly connected to the DAU) and takes care of broadcasting them to different sinks: Flight Attendant Panels (FAP) in the cabin, System Display (SD) in the cockpit, Ground Service Panel (GSP) for on ground video streaming outside of the aircraft, Digital Video Recorder (DVR) to record all camera streams on a removable storage device, Satellite Communication (SATCOM) for transmission of video and audio streams to a ground station during flight.

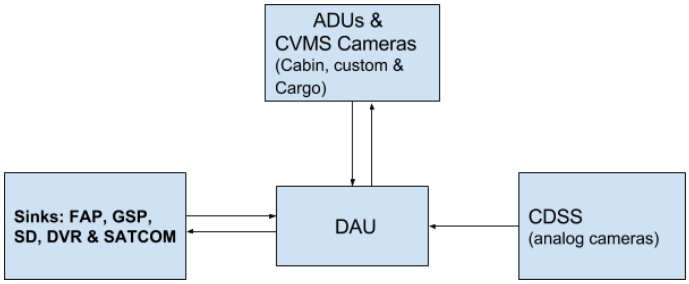



FIGURE 3.1: CVMS components involved in the video streaming function

When the DAU receives an http request from a sink, request is decoded to retrieve camera number and video format (single or quad).
If the request is coming from a FAP, CMV or SATCOM, the DAU checks for video display authorization by querying the REDIS database.
If the video is not allowed to be displayed on the sink, a black image with an “access denied test is sent”.
If the request is coming from GSP or Front Plug, video stream is directly retrieved and sent.

To implement these video management functions, the DAU runs a streaming server to handle requests from sinks and a streaming client to receive video streams from cameras.

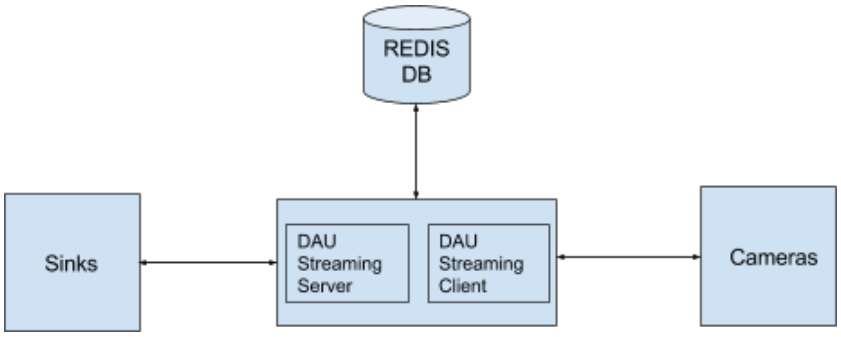



FIGURE 3.2: DAU a streaming server and client

### 3.1.1 Scope on the streaming server in SD mode

The CVMS DAU streaming web server provides video streams and a navigation HMI to FAPs and GSP. Videos are embedded in web pages at a typical frame rate of 15fps that can vary according to the lighting conditions. A JAVA-applet is implemented to play videos on a software media player at a resolution of 640 x 480 pixels for single view streams and 320 x 240 pixels for each stream for quad view (the size of the visualized frame of four streams grouped together has the same size of the one in single view). The bandwidth of streaming in Quad view is in average 8.5 Mbit/s and in single view is in average 5.4 Mbit/s.

SD cameras and DAU streaming server use MJPEG based video compression which is basically a stream of JPEG images transmitted over http.
The MJPEG codec corresponds to the definition IETF RFC 2435 and is parameterized to support the resolutions 640 x 480 and 320 x 240 pixels. The MJPEG stream sent over http looks like:

```
HTTP/1.1 200 OK
Content-Type: multipart/x-mixed-replace;
boundary=myboundary

 --myboundary

Content-Type: image/jpeg
Content-length: 12345
[image 1 encoded jpeg data]

 --myboundary

 Content-Type: image/jpeg
 Content-length: 45678
 [image 2 encoded jpeg data]

 [...]
```

FIGURE 3.3: MJPEG stream

### 3.1.2 Software used to play videos on Flight Attendant Panels and System Display

CVMS provides the JAVA-applet that will start the software video player installed on the FAPs and System Display for video visualization. The software's been fixed

in the Airbus technical specification to be Mplayer. The versions of the software depend on the FAP hardware version.

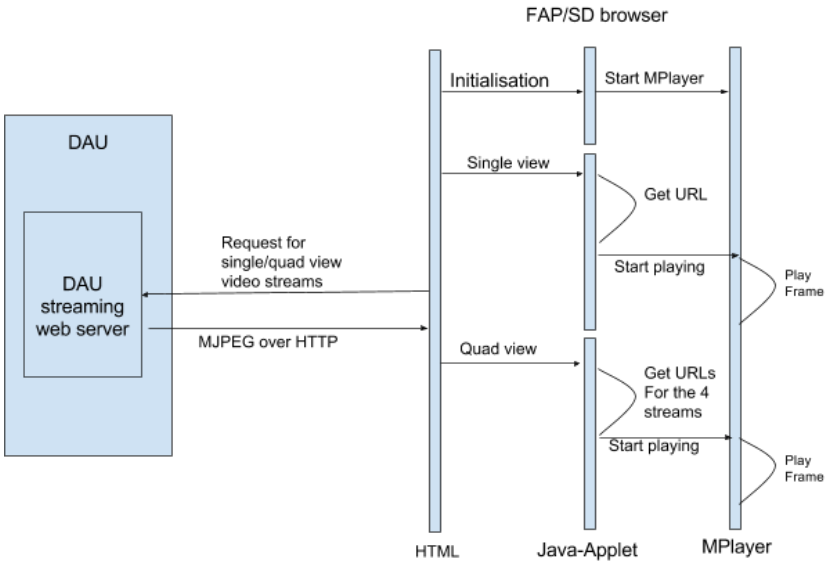



FIGURE 3.4: Java-Applet and MPlayer for displaying videos

**Overview on MPlayer**

An understanding of the main functions of the Media Player software is necessary in order to be able to import possible optimizations on the video streaming function of the system. The two basic MPlayer software components that intervene when playing a video stream are the : Demultiplexer and the Decoder. The demultiplexer takes care of splitting different multimedia objects interleaved in the stream (audio and video in our case ). It is also responsible for computing the decoding and the presentation time of every single frame. The decoder on its side takes care of decompressing and reversing the encoding of multimedia data before it is ready for playback.

When MPlayer is launched, the first thing it does is to analyze multimedia streams in order to find out :

- The demuxer to be used (depending on the video format)
- The decoder (depending on the codec, examples: MJPEG, Theora, MP4, etc)
- The video resolution
- The frame rate
- etc

```
zodiac@debian6:~$ mplayer -slave -idle -quiet -nolirc -nojoystick -nomouseinput -noaspect -vo x11 -loop 0 -nosound -framed
-vc +ffmjpeg http://172.22.250.14:8280/video?22
MPlayer 1.0rc3-4.4.5 (C) 2000-2009 MPlayer Team

Playing http://172.22.250.14:8280/video?22.
Resolving 172.22.250.14 for AF_INET6...
Couldn't resolve name for AF_INET6: 172.22.250.14
Connecting to server 172.22.250.14[172.22.250.14]: 8280...
Cache size set to 0 KBytes
Stream not seekable!
Forced lavf raw MJPEG video demuxer
libavformat file format detected.
[lavf] Video stream found, -vid 0
VIDEO:  [MJPG]  640x480  0bpp  25.000 fps    0.0 kbps ( 0.0 kbyte/s)
==========================================================================
Forced video codec: +ffmjpeg
Opening video decoder: [ffmpeg] FFmpeg's libavcodec codec family
Unsupported PixelFormat -1
Selected video codec: [ffmjpeg] vfm: ffmpeg (FFmpeg MJPEG)
==========================================================================
Audio: no sound
Starting playback...
VDec: vo config request - 640 x 480 (preferred colorspace: Planar 422P)
Could not find matching colorspace - retrying with -vf scale...
Opening video filter: [scale]
VDec: using Planar 422P as output csp (no 1)
Movie-Aspect is undefined - no prescaling applied.
[swscaler @ 0x8a1b280]using unscaled yuv422p -> rgb32 special converter
VO: [x11] 640x480 => 640x480 BGRA
Shared memory not supported
Reverting to normal Xlib

Exiting... (Quit)
zodiac@debian6:~$
```

FIGURE 3.5: MPlayer stream analysis

The figure above shows the results MPlayer gets after analysing the video streams it received from one of the CVMS cameras: No specific demultiplexer precised ( lavf raw MJPEG demuxer is used when raw encoded data is streamed without being encapsulated in any multimedia container ), the codec is MJPEG, video resolution is 640x480 and frame rate is at 25 fps.

## 3.2 Issues With The Current Video Streaming Transport Layer

Merely transmitting JPEG images as soon as they are received from cameras , is certainly a simple way of delivering MJPEG streams and getting the videos displayed and reordered correctly by different sinks. However some issues were noticed, mainly :

- A latency at the beginning of each new video stream being played. This is mostly annoying when changing from one camera stream to another on the FAPs.
- A visible time lag between what is filmed and what is displayed.
- The video frames are transmitted without any timestamp indicating the corresponding capture time. When playing them in real time there is no significant side effects. But recording them with DVR and then trying to play them back, there is no way to know the exact time for playing each frame.
- The frame rate of videos is assumed to be constant whereas CVMS cameras do not always capture streams at a fixed frame rate of 15 fps. In fact, CVMS cameras use two algorithms to control imager dynamic which are the Auto Exposure Control AEC and the Auto Gain Control.These algorithms control the sensor exposure time and a gain factor in order to allow image capture under variable light conditions in the aircraft cabins from 0.1 lux to 50,000 lux. Though, in extremely poor illumination, it is rather difficult to achieve a correct image brightness and a frame rate of 15 fps. In this case, CVMS cameras

activate the night mode by slowing down the original frame rate and waiting for a maximum exposure time defined by the system configuration. Setting the maximum exposure time to 200ms, allows for a frame rate slow down to 5 fps.

## 3.3 Goal and expected results

The goal is to analyze the system behavior in order to identify the main reasons behind the issues mentioned above. Based on these reasons, the current streaming transport layer needs to be modified and optimized.

The modifications to be made on the streaming server are mainly expected to:

- Eliminate the latency when a new video stream is requested to be visualized by the Flight Attendant Panel.
- Improve the synchronization between what is being filmed and what is being visualized.
- Handle a video streaming with a variable frame rate.
- Be as simple as possible and do not lead to a large increase of the streaming server's use of CPU resources.
- do not add a significant data overhead.

It is also required that the proposed solution modifies as few software components as possible. In case the solution is proved to be efficient, it is necessary to integrate it to the system while conserving compatibility with all the equipments that were designed according to the current streaming transport layer.

# Chapter 4

# Proposed Solution For Video Streaming Optimization

## 4.1 Video Streaming Multimedia Containers And Codecs

In order to guarantee an efficient multimedia transmission, consumption and storage, many challenges are encountered. A first challenge, is having the adequate codec with the required compression ratio and content quality. For video surveillance, a powerful codec is one that facilitates motion detection and object tracking and that allows efficient video storage. A second challenge, is putting the compressed content in a convenient form allowing it to be retrieved easily and rapidly without loosing quality.

### 4.1.1 Codecs

A codec is a computer program which on one hand encodes/compresses data allowing a fast and easy transmission, and on the other hand decodes/decompresses data when it is played back.

- lossy compression: Certain informations are eliminated in order to reduce the size especially redundant data, some frames in a video file, sounds barely heard in an audio file.
- lossless compression: Only redundant data is eliminated. Encoded data is fully restored at decode time.

A trade-off should between file size and quality should be made when choosing the compression method.
The codec used for CVMS video streaming in SD mode is MJPEG. This might be disadvantageous because MJPEG intra-frame video compression, hence comparing with other available methods it doesn't allow a significant size reduction. This is inconvenient when storing videos with the DVR module.

### 4.1.2 Multimedia containers[2]

A container is basically a file format encapsulating different compressed multimedia objects, allowing their identification and interleave. Hence, a multimedia container adds another layer before transmitting raw compressed data: It handles its packaging, transportation and presentation. Ogg and MKV provide advanced functionalities. They can support audio, video, subtitles, chapters, and metadata and a number of synchronization information needed for playback.
**Many multimedia containers are available. The most standard ones are:**

- Ogg, standard container for Xiph.org codecs: Theora, Vorbis.
- Matroska, it not standard for any codec.
- AVI, standard container for Microsoft Windows.
- MOV, standard container for QuickTime.
- MP4, standard container for MPEG-4.
- ASF, standard container for Microsoft WMA and WMV.
- RealMedia, standard container for RealVideo and RealAudio.

**To compare multimedia containers we basically consider the following criterias:**

- Popularity: is it a very commonly used container.
- Does it support multimedia streaming.
- Overhead: for the same raw data, each container adds a different amount of data depending on its internal structure
- Support for codec B-frames and Variable Bit Rate audio
- Support for chapters, subtitles, etc.

TABLE 4.1: Container formats Comparison

| Container | AVI | MP4 | OGG | MKV | ASF |
|---|---|---|---|---|---|
| **Owner** | Microsoft | MPEG | Xiph.org | Matroska.org | Microsoft |
| **Open Source** | No | Yes | Yes | Yes | No |
| **B-Frames support** | tweaks | Yes | Yes | Yes | Yes |
| **VBR audio** | Yes | Yes | Yes | Yes | Yes |
| **Variable frame rate** | No | Yes | Yes | Yes | Yes |
| **Chapters** | No | NO | Yes | Yes | Yes |
| **Stream-able** | No | Yes | Yes | Yes | Yes |

## 4.2 Optimizing Video Streaming Using A Multimedia Container

### 4.2.1 Which multimedia Container To Choose?

An open source multimedia container is needed in order to freely use it and implement it. Ogg has been chosen as it is the most well-known container in open source and it is more suited for live streaming than Matroska and MP4. A quite clear documentation for the format is provided in the Xiph.org website https://xiph.org/ogg/
.

The difficulty in streaming MJPEG into an Ogg bitstream is that it is not the standard way of streaming MJPEG over Http neither the standard way of using the Ogg multimedia container. Thus, there are no standard specifications provided for that. For that, it is needed to investigate the way MPlayer (as it is the media

player used in our case) identifies and decodes an Ogg bitstream. The source code of Mplayer was downloaded and logging messages as required were added in order to understand the software behaviour. This will be particularly necessary for the code related to the ogg multiplexer and the MJPEG decoder. VLC will be used to play the role of the streaming server ( which is the DAU in the CVMS system).

### 4.2.2 Streaming MJPEG into Ogg Format Over HTTP

#### 4.2.2.1 Ogg overview[3]

As a multimedia container, Ogg allows the encapsulation of raw compressed data and the multiplexing of independent multimedia streams for video, audio, text and metadata. Ogg is originally designed to be used with the Xiph codecs like Theora and Vorbis. Though it can also be used in a non standard way with other codecs like in our case the MJPEG. This is made possible because Ogg was designed without any restrictions on the content it carries, it doesn't even know anything about the data it encapsulates. It is left to every codec, to declare and identify itself. A particular feature of Ogg is that it also provides packet framing, error detection and periodic timestamp for seeking. The basic unit in an Ogg stream is an Ogg page. Each page is basically a header that is followed by a payload of a maximum of 255 data segments. One segment is at most 255 bytes which makes the size of an Ogg page limited to a maximum of 65307 bytes. As the figure below shows, an Ogg page header is composed of 8 fields that make a total of 27 bytes and a segment table containing the lacing values which are the sizes of all the segments in the page payload.

TABLE 4.2: Ogg Page Header

| Field | Size(bits) | Description |
|---|---|---|
| **capture_pattern** | 32 | magic number: Oggs |
| **version** | 8 | always zero |
| **flags** | 8 | |
| **granule_position** | 64 | abstract time-stamp |
| **bitstream_serial_number** | 32 | elementary stream number |
| **page_sequence_number** | 32 | incremented by 1 each page |
| **checksum** | 32 | CRC of entire page |
| **page_segments** | 8 | length of $segment_table$ |
| **segment_table** | variable | list of packet sizes |

Headering allows for a logical segmentation of the bitstream and avoids packet re-assembly (Which is necessary when physical segmentation of packets is used ). When encoding an Ogg bitstream, packet data is buffered, once the buffer size is enough to make a page, the header is written followed by a payload of the buffered data. To check for data loss, it is enough to compare the number of bytes got with the expected number which computed by summing the lacing values. The integrity of data is checked using the checksum field of the header. The field reserved for the stream serial number allows for the interleaving of different Ogg bitstreams. When decoding data, different bitstreams will be distinguished by the serial number of the page header.

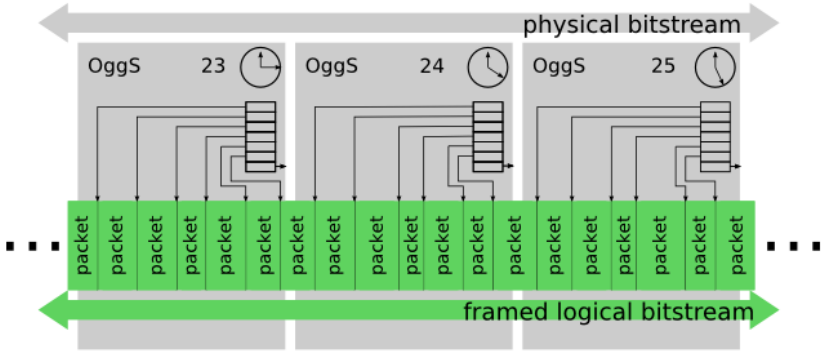



FIGURE 4.1: Ogg bitstreaming

#### 4.2.2.2 Embedding MJPEG into an Ogg bitstream

- **MIME type:**
  MIME instruction in the beginning of data transmission, allows the receiver to easily categorize the data and identify how it should be opened and read. There are different MIME types depending on the data carried by the container. Generally, MIME types related to the ogg container are: audio/ogg, video/ogg; application/ogg, etc. Though, using VLC to stream MJPEG into ogg and MPlayer to read the streams, then investigating the http packets on wireshark, we notice that the http response of VLC upon a request from MPlayer looks like:

```
 HTTP/1.1 200 OK
Content-Type: application/octet-stream
```

  We will as a result use application/octet-stream as the content-type even if it is defined as "arbitrary binary data" in RFC 2046.

```
09 65 f8 fd 48 54 54 50  2f 31 2e 30 20 32 30 30   .e..HTTP /1.0 200
20 4f 4b 0d 0a 43 6f 6e  74 65 6e 74 2d 74 79 70    OK..Con tent-typ
65 3a 20 61 70 70 6c 69  63 61 74 69 6f 6e 2f 6f   e: appli cation/o
63 74 65 74 2d 73 74 72  65 61 6d 0d 0a 43 61 63   ctet-str eam..Cac
68 65 2d 43 6f 6e 74 72  6f 6c 3a 20 6e 6f 2d 63   he-Contr ol: no-c
61 63 68 65 0d 0a 0d 0a                            ache....
```

FIGURE 4.2: HTTP response packet from VLC to MPlayer

- **Bitstream headers:**
  Each Ogg bitstream header is transmitted in a dedicated page. The number of headers depends on the codec. Generally, at least one page header is transmitted at the beginning of the stream and before transmitting any data packets. The first page header usually includes informations about the type of codec, the time base, the time unit, the video resolution (height and width), the buffer size, etc. Any other informations needed for the codec set up might be also included. The mapping of the header fields needs to be specified in the codec as no standard mapping is defined in the Ogg documentation.

- **Bitstream data packets:**
  Data packets carry the data to be decoded on the receiver side. In case of MJPEG streaming, the Ogg bitstream data packets will encapsulate the JPEG images.

#### 4.2.2.3 Packet Timestamp and Variable Frame Rate Considerations[4]

**4.2.2.3.1 Variable Frame Rate Definition** CVMS cameras produce videos at a variable frame rates. In the best lighting conditions the video streaming is at 15 fps, while in the worst case, it is at 5 fps. This variation messes up what frame belongs to what portion of video.
The image below shows what actually happens in real time capture, the delay between frames isn't constant, and we may also skip some frames

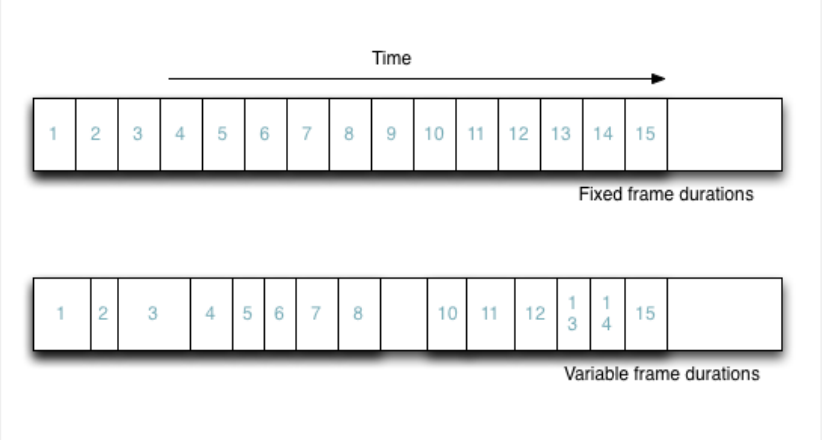



FIGURE 4.3: Fixed Frame Rate VS Variable Frame Rate

**4.2.2.3.2 Issues Related To Variable Frame Rate Videos**

- **Video and Audio get out of synchronization**
  In general the most common video editing software do not support a variable frame. And this leads mainly to a problem in the synchronization between audio and video. In fact the audio will never be affected by lighting conditions, while the video frame rate can vary significantly. As a result, if the video player software continues to play the video and audio at a constant frame rate, they will lose their synchronization and this can be very annoying.

- **Time drift when playing recorded video:**
  Another problem is also seen when playing the recorded video. In fact because of the variable frame rate, sometimes the video will be played too fast and the time is not correct. Hence playing a video recorded at a variable frame rate as a constant frame rate video results in a time drift.

**4.2.2.3.3 Ogg Format Support For Variable Frame Rate** When streaming the CVMS MJPEG video streams, no Timestamps are carried. We only deliver a series of JPEG images as fast as they are captured by the cameras. So Mplayer, specifically ffmpeg, will need to make up timestamps to process the Frames. This is done by continuously counting the received frames over a second then comparing the result with the expected value. If they are the same, nothing should be done, if they are different two situations can occur: either we got less frames than we expected in this case the missing frames will be compensated by duplicating some of the received frames, or we got more frames than expected in this case some frames will be dropped. Mplayer allows us to specify the desired frame rate by fixing the parameter -fps to a constant value. Using this value, MPlayer computes the delay between frames as

1/fps Some container formats do not really support variable frame rate, but they can "fake" it somehow. For example, AVI gets the highest common multiple of all the possible frame rates and use it to play the video, some frames will be duplicated in order to fill in the gaps. Other multimedia containers like Matroska and Ogg completely drop the idea of frame rate and set an individual timestamp for each frame. We will focus on Ogg as it is the container we decided to use for our system.

**4.2.2.3.3.1 Ogg Timestamp**

Each Ogg page (not packet) header contains a timestamp, or granule position in Ogg terms, encoded as a 64-bit number. The precise interpretation of this number is not defined by the Ogg specification; it depends on the codec used for each elementary stream. The specification does, however, tell us one thing: Each Ogg page has in its header a 64 bits field reserved for the Timestamp. In Ogg terms the timestamp is called the granule position of the page. The specification does not tell us how to compute and interpret the granule position. It is up to each codec to know how to encode the timecode it expects using the 64 bits. This gives the Ogg container more flexibility and allows for an absolute time stamping instead of a hard coded one. However, one thing should be put in mind when computing the granule position is that:
*"The position specified is the total samples encoded after including all packets finished on this page (packets begun on this page but continuing on to the next page do not count)."*

A sample is a video frame for video (an image) and a PCM sample for audio.

**4.2.2.3.3.2 Timestamp interpretation for MJPEG streaming**

As mentioned previously, the meaning of the 64-bit timestamps associated with an elementary stream depends on the codec of the stream. A lot of tricks can be played in order to take advantage of the granule position field of the Ogg container. Each code will define a specific mapping depending on whether it uses inter or intra frame encoding. A key point that should be taken into consideration when computing the granule position is how it is used, at the media player software, to get the presentation timestamp and the decompression timestamp.

MPlayer defines these values as:

- **Presentation timestamp PTS:** the time at which the decompressed packet will be presented to the user. Can be $AV\_NOPTS\_VALUE$ if it is not stored in the file.
- **Decompression timestamp DTS:** The time at which the packet is decompressed. Can be $AV\_NODTS\_VALUE$ if it is not stored in the file

As a frame needs to be decompressed before being presented,the presentation time always needs to be larger than the decompression time. PTS and DTS values were introduced mainly to deal with inter frame codecs as frames are dependent on each other, and one frame can depend on an earlier frame.

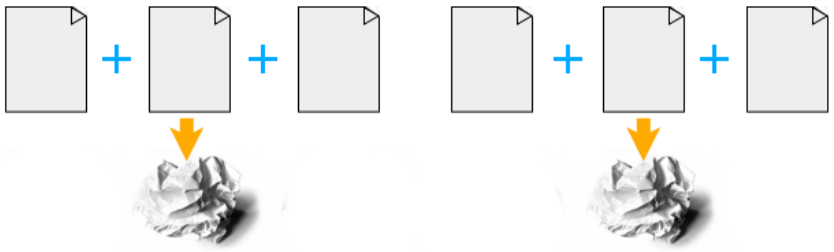


FIGURE 4.4: Inter-frame compression

To better understand this, inter frame codecs define 3 types of frames:

- I-frames are the least compressible but don't require other video frames to decode.
- P-frames can use data from previous frames to decompress and are more compressible than I-frames.
- B-frames can use both previous and forward frames for data reference to get the highest amount of data compression.

for example if a frame is to be displayed as I B B P, our stream might look like:
**PTS** : 1 4 2 3
**DTS** : 1 2 3 4
**Stream** : I P B B
As a concrete example of an inter frame codec, we can mention Theora which splits the 64-bit timestamp into two fields. The first of these fields encodes the frame number, starting from one, of the key frame most recently preceding the frame to which the timestamp applies. The second timestamp field encodes the number of frames, this time starting from zero, since the most recent key frame. The bit position of the field split is specified by the Theora stream header.

Now in case of the MJPEG codec, things are much simpler as it is an intra frame codec, i.e. data of one single frame is analysed in order to compress independently from any other frame.

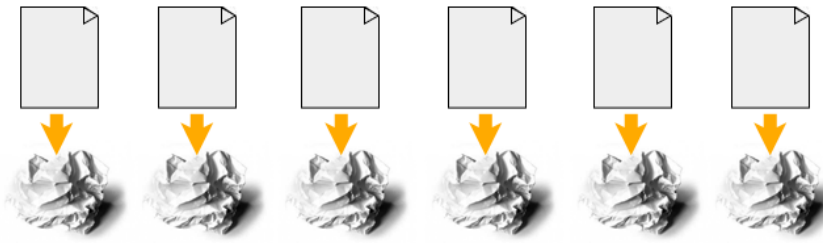


FIGURE 4.5: Intra-frame compression

As a result, for MJPEG the PTS and DTS timestamps are equal and frames can be decoded and presented in the same order they are streamed. For the granule position it can simply be the position of the frame in the stream counting from the first streamed one.

The unit of the timestamps is defined in the first Ogg page (The header page) as 1/fps multiplied by a factor of 10000000 (following the value used by MPlayer ) which allows for a better precision.

# Chapter 5

# Proposed solution implementation and Development

## 5.1 Work environment

### 5.1.1 System Test Bench Description

The system test bench is realized in a way to simulate as much as possible the real system in the aircraft. The DAU, CVMS cameras and analog cameras are the same as the ones used on board. For the CVMS sinks they are simulated with: A normal computer for the Ground Service Panel which is accessed via a web browser. For FAPs they are simulated using an ISO file delivered by Airbus and run in a virtual machine. All the CVMS sink simulators need to be connected to the same local area network as the CVMS. ADUs are replaced with normal network switches.

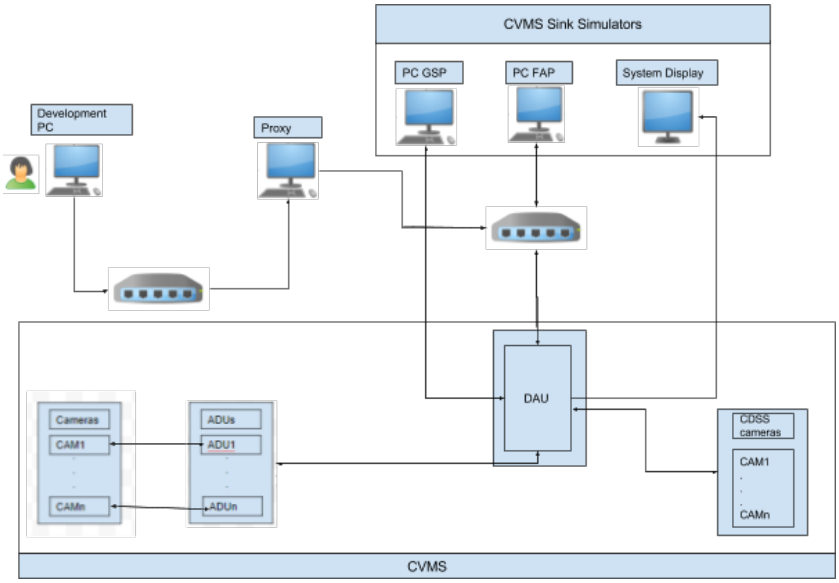



FIGURE 5.1: CVMS System Test Bench

The system Test bench is used in the software development phase in order to easily make tests.

### 5.1.2 Code development and testing

A PC with a Linux OS for code development. I used eclipse to edit the code.
Once I need to test my changes I only need to generate a new software image using PTxdist, then I use ssh tools to send the image to the external SD card plugged to the

DAU. However changing the software image in the SD card doesn't make the DAU SOCs update their softwares. One possibility is to manually invalidate the software of each SOC then to reboot them all. To test the video visualization, I connect my machine using ssh to the virtual machine emulating a FAP and I launch MPlayer. To automate ssh connections to different test bench components, the following configuration is used:

```
Host pctest?vm
User zodiac
ControlMaster auto
ControlPath /tmp/ssh-%r@%h:%p
ControlPersist 30
ServerAliveInterval 10
ForwardX11 yes
Host pctest1vm
Hostname 192.168.1.200
Host pctest2vm
Hostname 192.168.1.201

Host msoc*
User root
IdentityFile ~/.ssh/id_rsa_dau
ProxyCommand ssh pctest1vm nc -w 2 %h %p
#ProxyCommand ssh pctest2vm nc -w 2 %h %p
#ProxyCommand ssh cyber3.zodiac-ife.info nc -w 2 %h %p
StrictHostKeyChecking no
UserKnownHostsFile /dev/null
LogLevel INFO
ControlMaster auto
ControlPath /tmp/ssh-%r@%h:%p
ControlPersist 30
ServerAliveInterval 10
Host msoc1
Hostname 172.22.250.11
Host msoc2
Hostname 172.22.250.12
Host msoc3
Hostname 172.22.250.13
Host msoc4
Hostname 172.22.250.14
LocalForward 22222 127.0.0.1:2345
Host msoc5
Hostname 172.22.250.15
Host msoc6
Hostname 172.22.250.16
Host msoc7
Hostname 172.22.250.17
Host msoc8
Hostname 172.22.250.18
```

*pctest1vm* and *pctest2vm* are two different virtual machines each one installed on

a different computer belonging to a different test bench. These two machines are used to simulate FAPs and to serve as a proxy to allow connections to DAU msocs.

## 5.2 Streaming MJPEG Over Ogg Demo

When working with open source software, it is not always guaranteed that it will behave as expected.
This is even more complicated when using open software in a non standard way as in our case. In fact, even if the Ogg multimedia container is known to be designed to be a general purpose container format which allows to encapsulate any type of encoded data, it is usually used only with codecs designed by Xiph like Theora and Vorbis. Instead, using it to encapsulate MJPEG encoded data there are no standard specifications that can be found. Hence, the first step through implementing the Ogg layer is to identify a clear mapping for the MJPEG codec. The mapping should define how to identify the codec type, how to compute timestamps for packets , which set-up data are needed and how to extract them. It should also be determined based on the elements expected by MPlayer to correctly decode Ogg streams. That's why before starting to work on the CVMS software, a simple HTTP video streaming server was developed to implement the Ogg layer. Working on a simple server allows to avoid the complexity of the CVMS system and to easily localize and debug errors. At this step, VLC was used as a media streaming software that can be seen as a reference since it is able to transmit MJPEG over Ogg streams that can be correctly decoded by MPlayer.
So to identify the key elements to develop the server, the traffic between VLC and MPlayer was analysed using Wireshark . The source code of these two softwares was also investigated. The server was built on top of an open source MJPEG streaming server on GitHub at the following link: https://github.com/ccrisan/streameye To develop the Ogg transport layer, the Xiph library Libogg at https://xiph.org/downloads/ was used.

### 5.2.1 Demo Streaming Server Overview

The tests on MPlayer and VLC allowed to conclude that MPlayer expects to receive at least on Header packet for codec identification and setup. For the JPEG data packets they are expected to be encapsulated in a specific way. More details in the descriptions that follow.

#### Ogg Stream Header Packets

Header packets are to be sent initially before sending any data packets. A header packet is by itself a regular Ogg packet. Which means it is composed of a header and a body. The header is the normal header of a regular Ogg packet. The body should be defined depending on the type of the Header packet. Looking at MPlayer's source code two header packets are streamed for MJPEG codec:

- **Codec Setup Header Packet**
  Its first byte needs to have a ZERO value. It is used to define the codec type and setup elements. The Header fields are defined as follows:
  - Packet type: One byte that is set to ONE to indicate that it is a header packet.

- Stream type: A string of 8 characters that indicate the type of the stream. For video streaming, it should be set to "video"
- Subtype: A string of 4 characters. In case of MJPEG streaming it should be set to "MJPG".
- Size: A 4 bytes integer storing the size of the packet in bytes
- Time Unit: An 8 bytes integer indicating the time unit that should be used by MPlayer to compute the presentation and decode time of frames.
- Samples per unit: An 8 bytes integer that can be simply set to 1.
- Default length: A 4 bytes integer, can be set to 1.
- Buffer size: A 4 bytes integer
- Bits per sample: 2 bytes integer, it is not clear how MPlayer makes use of it.
- Padding 0: 2 bytes for padding (to ensure a fixed size for the structure.)
- Ogg header video : structure containing 2 fields of 2 bytes each. One for the height and another for the width: allows MPlayer to know the video resolution.
- Ogg header audio : structure of 3 fields, one of 2 bytes for the audio channel, another of 2 bytes for the block alignment and another one of ′ bytes for the number of bytes transmitted per second.
- Padding 2: Another field of 4 bytes for padding.

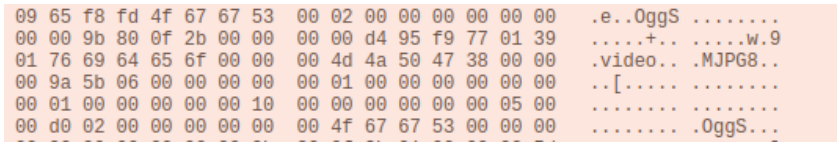



FIGURE 5.2: Ogg setup header packet

- **Comment Packet** Its first byte needs to have a value of 3. It can be used to send further informations on the video stream.

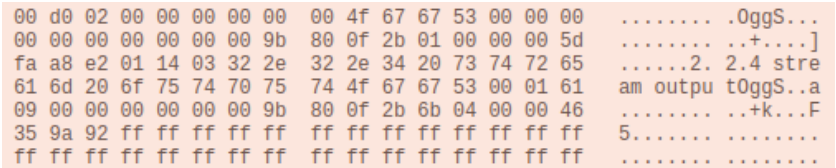



FIGURE 5.3: Ogg setup comment packet

### Stream Data Packets

The stream data packets are the ones encapsulating the video frames. Each time the server receives a JPEG image it encapsulates it in an Ogg packet. The Ogg packet is then submitted to the Ogg bitstream. Finally, to stream the sequence of images over Http, it is sufficient to extract Ogg pages from the bitstream one by one and send them to the network.

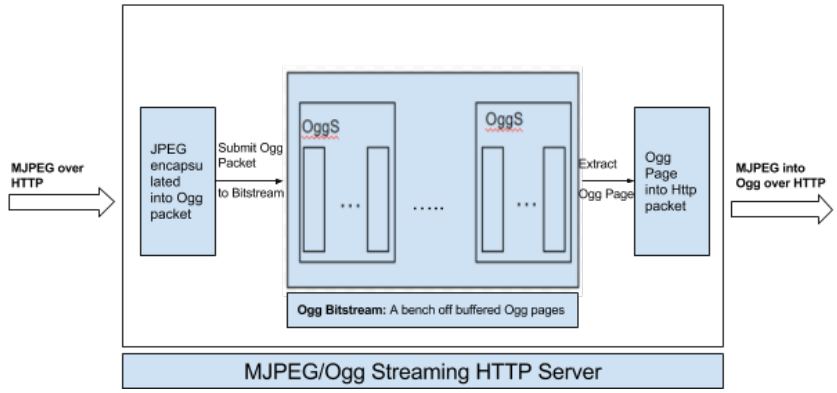



FIGURE 5.4: Ogg demo streaming

### 5.2.2 Demo Streaming Server Testing

The Demo server has also a client side that will receive the MJPEG streams. To emulate the MJPEG stream, a simple script was used.

```
webcam_emul.sh:
#! /bin/sh
while true; do
    for file in directory_storing_jpeg_images/*.jpg; do
        cat $file
        sleep 0.04
    done
done
```

The script will stream a bunch of JPEG images forming a coherent video. One possible way to get these images is to use MPlayer dumping tool to convert a video to a sequence of JPEG frames. mplayer -vo jpeg -sstep 1 video.avi : This will extract a video frame each 1 second

To make tests on the server, it is sufficient to launch the following command:

```
./webcam_emul.sh  | ./streameye/streameye -s
"--separator--" -p 8082 -l -d
```

## 5.3 Solution Implementation On The Data Acquisition Unit

The DAU software part that needs to be modified in order to implement the Ogg layer is the Http Streaming Server module.

### 5.3.1 Overview On The DAU Http Streaming Server

This module receives video streams HTTP requests from different sinks: FAP, CMV, Satcom, GSP, Frontplug. Each HTTP request identifies which camera video stream is needed and whether a Single View or a Quad View is required. Upon a video request, the server checks whether the sink is eligible for video delivery, if it is not it sends a Black/ video not available screen. For every connected sink, the server creates a corresponding thread: FAP Thread , CMV Thread , GSP/FrontPlug Thread. A sink Thread mainly functions as follows:

- Wait for a request from the corresponding sink:
  Each sink Thread creates a socket in nonblocking connected mode and call to select() system call for waiting to incoming request from sinks.

- Process request once received:
  When a socket receives a request from a sink, request is decoded to retrieve the camera number and view format (Single or Quad view) of the requested video stream. If request is coming from FAP, CMV or SATCOM, the server checks for authorization in the REDIS database by reading the Display Status for the corresponding camera. If video is not allowed to be displayed on sink a black image with on-screen text "Access denied" is sent. If request is coming from GSP or Front plug, video stream is directly retrieved and sent.

- Transmit video if authorized
  To retrieve and transmit video stream to sink, HSSM first gets the MediaProfile object corresponding to video stream in MediaProfile database using the token (String) indicating camera number and view format. With MediaProfile object, HSSM gets a MediaStreamController object.

### 5.3.2 Streaming Server Data Transfer Mechanism

The data transfer mechanism that is used is the chunked buffer encoding which is based on streaming HTTP messages of an unspecified number of chunks, a terminating chunk of length zero, a trailer possibly containing useful headers and a CRLF (carriage return and a Line feed).

Each chunk starts with the number of bytes it carries in it payload and possibly other parameters. The chunked encoding layer is to be considered when implementing the Ogg layer. The http chunk headers might in fact be inserted between the Ogg pages and they won't be removed by the http layer of MPlayer. As a result the Ogg demuxer might not be able to correctly decode the data packets.

### 5.3.3 Re-implementation of MediaStreamController services to support Ogg

MediaStreamController object provides three services:

- startStreams: to start video transmission from HTTP Streaming Client module
- stopStreams: to stop data transmission from Streaming Client module
- registerCallback: to send video stream data

To implement the additional multimedia container layer, it is sufficient to modify the services of the Media-profile object as follows:

- **Start Streams Service :**
  Before starting to transmit video streams to the Client, the Ogg page header should be transmitted. The header will include setup informations about the type of codec, the time base, the time unit, the video resolution (height and width).

```
Frame rate = 15, supposing a fixed frame rate of 15 fps.
Video width = 640;
Video height = 480
```

```
Packet_type = PACKET_TYPE_HEADER, looking to MPlayer code
it should be 1.
Stream type= "video"
Sub type = "MJPG"
Time unit = 10000000* 1/frame rate
Samples per unit = 1
Buffer size = 1024*1024
Bits  per sample = 0
```

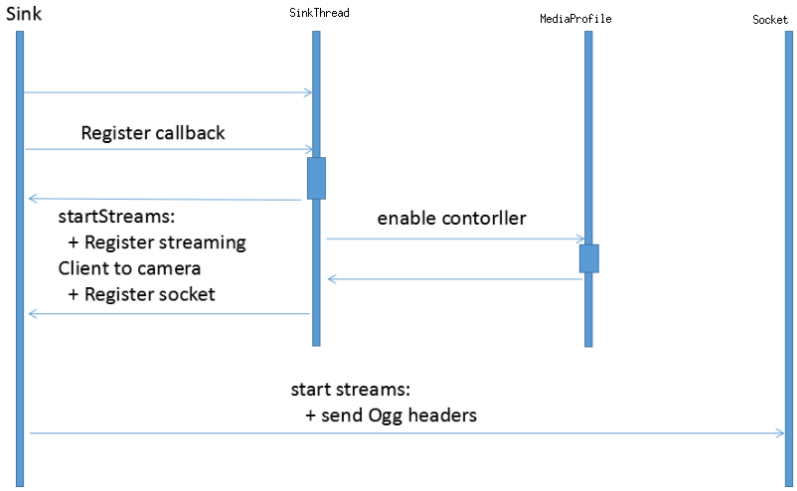



FIGURE 5.5: Ogg headers sent by Startstreams service

- **Register Call back Service:**
  The service is used to register a MediaStreamCallback object. Only one single method of the MediaStreamCallback object needs to be modified. This method is pushBuffer() and it is called by the HTTP Streaming Client module to send video streams. PushBuffer is re-implemented as follows:

  - JPEG image buffer is passed as a parameter to the function.
  - Encapsulate JPEG image into an Ogg packet. The header of the packet will store the size of the image, the position of the image in the video stream, a flag indicating whether it is the last or the first packet in the stream. The body of the packet will be a copy of all JPEG data initiated by the synchronization byte (=8). It is necessary to put this byte at the beginning otherwise MPlayer will not know that it is a data packet.
  - Submit the packet to the Ogg bitstream.
  - Extract all possible Ogg pages from the Ogg stream and concatenated them to form a complete JPEG image (which is now in a form of a sequence of Ogg pages, i.e it doesn't only contain its JPEG data but also the Ogg page headers )
  - Encapsulate the sequence of concatenated Ogg pages in a single chunk buffer. This is done in order to avoid the http chunk headers being inserted between the Ogg pages which leads to incorrect decoding of the video data.
  - Send Chunk Buffer.

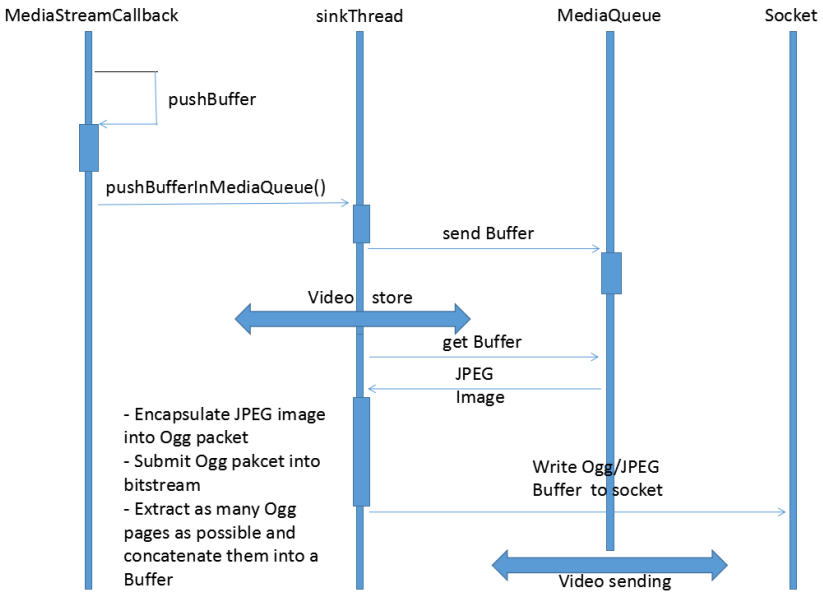



FIGURE 5.6: pushBuffer() method re-implementation to stream Ogg packets

# Chapter 6

# Performance analysis and comparisons

## 6.1 Stream Analysis Duration

As discussed before when MPlayer is launched, the first thing it does is to analyse multimedia streams in order to find out :

- The demuxer to be used (depending on the video format).
- The decoder (depending on the codec, examples: MJPEG, Theora, etc).
- The video resolution.
- The frame rate.

In case of MJPEG streaming, the Lavf demuxer is identified as soon as MPlayer is Launched. Lavf then starts to analyse video streams to identify the adequate codec. Since MJPEG is just a sequence of raw JPEG encoded data, the stream analyse duration is relatively quite long.
As the demuxer analyses the stream, more and more data gets buffered. Once the codec is identified, the buffered data will be decoded and displayed at a frame-rate of 25fps hardcoded in MPlayer's source code. This is has two impacts on the video visualization on FAPs:

- An noticeable effect of acceleration on the video.
- A time lag between the filmed video and the one displayed.

In case of Ogg streaming, MPlayer immediately identifies that it consists of an Ogg stream and it delegates the stream analysis to the Ogg demuxer. The Ogg demuxer on its side identifies the MJPEG codec and its setup elements as soon as it receives the first Ogg header packet.

As a result, Ogg allows a very important optimization on the analysis duration of streams with respect to MJPEG.

## 6.2 CPU Usage

The video application uses more CPU resources in the case of MJPEG streaming in Ogg container. This is because the Ogg layer adds a significant number of memory writes and reads. An example of one significant write operation , is when Ogg pages are copied and concatenated in a common buffer to be delivered in a single chunk buffer . This is done in order to conserve compatibility with the chunked-buffer HTTP layer.

If we want to avoid concatenating Ogg pages and we decide to send each Ogg page on its own, MPlayer faces issues when decoding video streams due to the additional headers added on top of each HTTP chunk buffer. To compare the CPU usage between Ogg and MJPEG the following scenarios were tested:

- **One camera One client**

| CPU% | Minimum | Maximum | Average |
|---|---|---|---|
| MJPEG | 4.5 | 10.8 | 7 |
| Ogg | 9 | 14 | 11 |

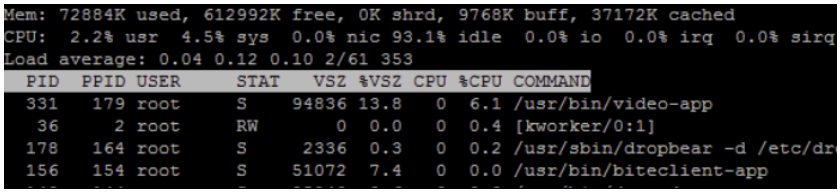



FIGURE 6.1: MJPEG streaming 1 camera 1 client

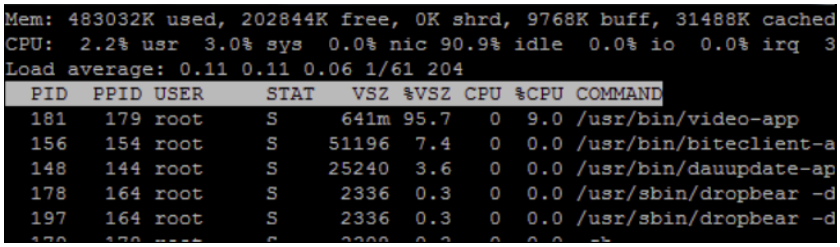



FIGURE 6.2: Ogg streaming 1 camera 1 client

- **One Camera Four Clients**

| CPU% | Minimum | Maximum | Average |
|---|---|---|---|
| MJPEG | 13 | 22 | 16 |
| Ogg | 18 | 30 | 22 |

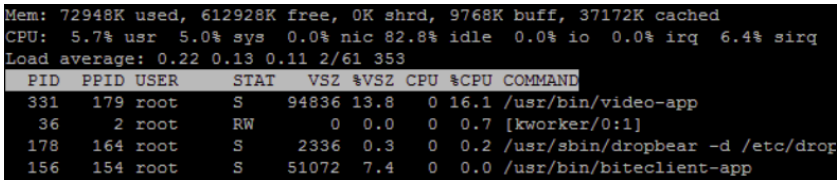



FIGURE 6.3: MJPEG streaming 1 camera 4 clients

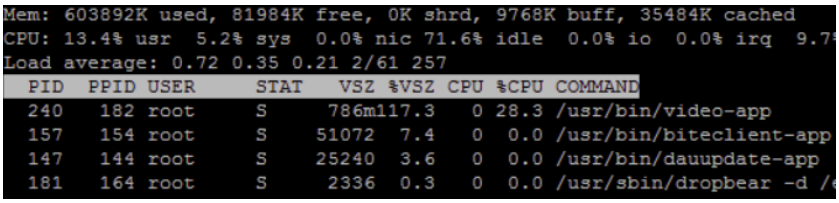



FIGURE 6.4: Ogg Streaming 1 camera 4 clients

- **Two Cameras Four Clients**

| CPU% | Minimum | Maximum | Average |
|---|---|---|---|
| MJPEG | 19 | 26 | 22 |
| Ogg | 27 | 36 | 32 |

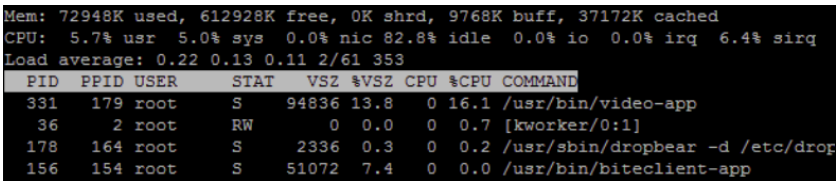



FIGURE 6.5: MJPEG Streaming 2 Cameras 4 Clients

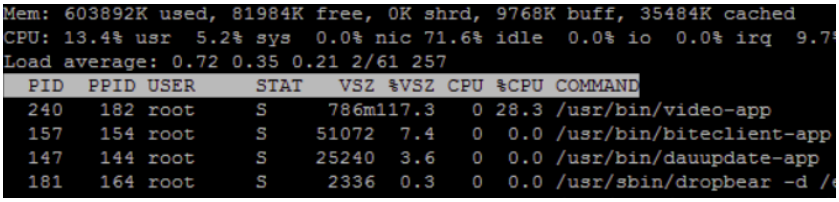



FIGURE 6.6: Ogg Streaming 2 Cameras 4 Clients

## 6.3 Performance at a Variable Frame Rate

MJPEG simply streams JPEG images without including any metadata. Ogg instead allows to timestamp video frames before streaming them. This might be of a minor importance when reading video streams in real time. As in both cases, MPlayer will display images as soon as it receives them. Though, timestamping video frames could make significant improvements when reading the CVMS DVR recorded videos. In fact, as it's been discussed before CVMS videos do not really have a fixed frame rate as it might change depending on the cabin lighting conditions.
And as videos get recorded with no timestamp the frames, the media player software will have no idea one when to present each of them and will read them at a fixed frame rate. An acceleration effect will hence be viewed.

As the current DVR was designed to receive an MJPEG stream, it is possible to conserve compatibility by providing the Ogg stream all together with the MJPEG stream. The stream format to be forwarded will be indicated by the Sink in the HTTP request along with the camera number and the view type ( single/quad).

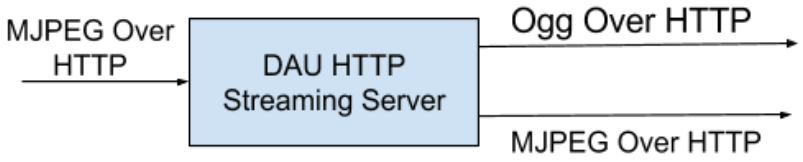



FIGURE 6.7: Ogg support while conserving MJPEG streaming

## 6.4 Data Overhead

One consideration to take into account is the additional data needed for the Ogg layer with respect to MJPEG. The overhead added by the Ogg layer has two types: a fixed overhead of 27 bytes for the Ogg page header fields and a variable overhead that depends on the Image size. The fixed overhead is relatively negligible. The variable overhead comes from the segment table fields which lists the sizes of all segments in the ogg page. Each value is one Byte and the number of values is 255 maximum. The packet size is simply the sum of all these values. Thus the Ogg overhead contributes with 1/255 or nearly 0.4% of the packet size . This overhead is much higher than in case of other multimedia containers like MP4 which has an overhead of only 0.04%. CVMS cameras stream images of an average size of 60k . Hence, the encapsulation of each image in an ogg packet makes an overhead of $60k/255 = 235$ bytes. As a result, in normal light conditions and when the frame rate is 15 fps, we will have an overhead of around 4 kilobytes each second. This amount of data is quite acceptable and doesn't lead to any noticeable side effects.

## 6.5 Latency

Ogg introduces some latency at the streaming server side and at the MPlayer side. A full Ogg page can be transmitted only when the entire content is buffered as the page header depends on the packet size and checksum. This makes the lower bound for latency equivalent to the duration necessary to buffer complete page.

On the MPlayer side, it can start decoding only after receiving and reassembling of all the pages belonging to the same video frame (One packet may span more than one page). Thus, the overall latency summing the one at the sender side and at the receiver side is approximately twice the duration of one page buffering.

One way to minimize latency is to minimize page duration by sending one packet per page. But in this case we increment the Ogg overhead, as the number of pages, thus of Ogg page headers, will increment.

# Conclusion

The Zodiac A380 surveillance system provides more functionalities than the previous system designed by Parker Hannifin. The key improvement it makes is that of streaming in both Simple and High definition technologies. Though, in SD mode the system doesn't appear to perform as good as in HD mode. Implementing the SD transport streaming layer as an MJPEG stream carried over HTTP, is certainly the simplest way to transmit JPEG images captured by surveillance cameras to different CVMS sinks.
Though, adding one more layer on top of MJPEG to encapsulate raw encoded data into a multimedia container allows for significant improvements. This will for sure add an extra overhead of data and increase the streaming server's CPU usage. But based on the tests that were made, this shouldn't have any particular side effects on the system performance. Ogg container was proved to be of minor complexity. A complete library is made available on the Xiph web site and implements all the basic functions of the standard. The integration of the Ogg layer to the CVMS system is rather simple as few software components are to be modified. However, even if Ogg is to be implemented, it is still necessary to keep providing MJPEG streams in order to conserve compatibility with the CVMS equipments that were already designed to receive MJPEG streams.

# References

[1] $http://www.zodiacaerospace.com/fr/groupe/presentation$
[2] $http://www.divxland.org/en/article/15/multimedia_container_formats\#.WS5qTPnyiM8$
[3] $https://www.xiph.org/ogg/$
[4] $https://support.shinywhitebox.com/hc/en-us/articles/200134755--Variable-Frame-Rates-Explained$
[5] $http://stackoverflow.com/questions/14988910/correct-mime-type-for-multipart-mjpeg-stream-over-http$
[6] $https://www.theora.org/doc/Theora.pdf$
[7] $https://people.xiph.org/\ giles/2004/openweekend/theora-talk/theora.pdf$
[8] $https://www.bunkus.org/videotools/ogmtools/$
[9] $http://www.streamplug.info/fr/encodage/ogm.html$
[10] $https://github.com/ccrisan/streameye$
[11] $http://techslides.com/sample-files-for-development$
[12] $http://www.viscoda.com/en/support/faqs/87-splitting-a-video-file-into-a-sequence-of-images-mplayer$
[13] $http://www.johnchow.com/an-easy-fix-for-editing-variable-frame-rate-videos/\#$
[14] $http://forum.doom9.org/archive/index.php/t-156236.html$
[15] $http://hardwarebug.org/2008/11/17/ogg-timestamps-explored$
[16] $http://dranger.com/ffmpeg/tutorial05.html$
[17] $https://en.wikipedia.org/wiki/Time_base_correction$