Hellaynnea Consulting
hellaynnea@gmail.com

NetMultiPlayer

Policy-Based Networking

(Ref.: NMP_NETPOL_ YL)

April 10th, 2006

Table of Content

Introduction

The object of this document is to describe the global policy of bandwidth management in a TCP/IP remote-monitoring network composed by:

- Recorders (black boxes)
- NetMultiPlayer video-camera monitoring software
- NetMultiPlayer Server, MPEG-4 streams router.

The NetMultiPlayer architecture already uses bandwidth management mechanisms thanks to recorder's Video Daemon, a powerful MPEG-4 encoding and streaming engine.

References

- The software development plan (SDP) of the Net Multiplayer software system. This document gives a detailed schedule of the project workload.

- The software architecture document (SAD) of the NetMultiPlayer software system. This document describes the full architecture of the project and interactions between the different pieces of software involved.

Introduction to MPEG4

Remark:

The following text has been taken from different websites giving documentation of MPEG-4. It is not part of my work and is just provided to help the reader have a better understanding of my work.

References:

http://www.openmux.com/mambo/Public/mpeg-4.pdf

http://www.scs.org/scsarchive/getDoc.cfm?id=2882

What Is MPEG-4?

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group). The first version of the MPEG-4 standard was finalized in October 1998 and became an international standard at the beginning of 1999. Although defined as one standard, MPEG-4 is actually a set of compression/decompression formats and streaming technologies that address the need for distributing rich interactive media over narrow and broadband networks.

Although the MPEG-4 architecture provides different "cores", the NetMultiPlayer, SipMultiPlayer and Video Daemon programs are only exploiting the MPEG-4 Visual technology specifying the representation of natural and synthetic visual objects. The MPEG-4 Visual standard allows encoding of natural (pixel based) images and video together with synthetic (computer generated) scenes. It also supports the compression of synthetic 2-D and 3-D graphic geometry parameters (i.e. compression of wire grid parameters, synthetic text). MPEG-4 Visual supports encoding bitrate between 5 Kbps and 10 Mbps, with resolutions from QSIF to Full D-1.

MPEG-4 Frame Structure

MPEG video streams are constituted with three types of frames: Intra-coded (I), Bidirectional (B) and Predicted (P) frames regularly spaced in time at a given *frame rate*. I frames are coded independently of other frames using transform coding and provide an access point to the compressed data. P frames use motion-compensated prediction from the most recent previous I or P frame. B frames are coded based on both past and future I or P frames. As a result, the size of a B frame is approximately half the size of a P frame which in turn is one third the size of an I frame.

Frames are organized in small sets called Group of Pictures (GOP). A GOP is constituted with an I frame and the consecutive B and P frames before the next I frame. MPEG video streams are typically characterized by a regular and repetitive GOP pattern which can be characterized by two parameters: L representing the distance (in number of frames) between two consecutive I frames, and Q which identifies the distance (in number of frames) between an I frame and the first P frame inside a GOP. The existence of this regular and repetitive GOP pattern is not required by the standard but widely used in practice.

Glossary

Bandwidth

It represents the amount of data that can be transmitted in a fixed amount of time. For digital devices, the bandwidth is usually expressed in bits per second of bytes per second (*bps*). For analog devices, the bandwidth is expressed in cycles per second, or Hertz (Hz).

BPS

Bits per Second (see *Bandwidth)*

Streaming

It describes a technique for transferring data so that it is received as a continuous real-time stream. Streaming refers mainly to audio and video data, which is time-dependent. Video files, especially, are very large and cannot be downloaded easily by home Internet users. Streamed data is transmitted by a server application and received and displayed in real-time by client applications. These applications can start displaying video or playing back audio as soon as enough data has been received and stored by the application.

GOP

Standing for *Group of Pictures*, it describes a set of MPEG-4 frames (I, B or P). The parameters of the GOP characterize the way the frames will be organized inside the set.

MPEG-4 Encoding With Adjustable Bitrate

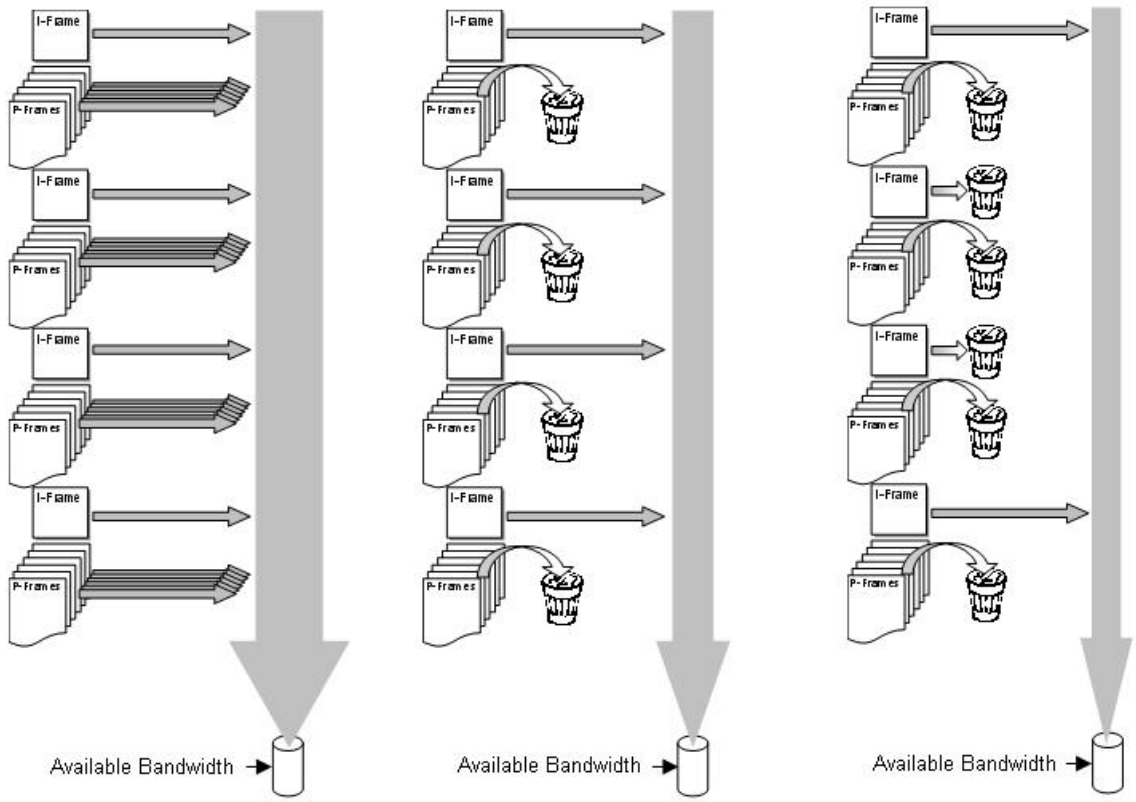
Adjustable Constant Bitrate Streaming

When a user needs to access to a video-stream, it sends a request to a recorder's Video Daemon indicating the desired bitrate value.

Example:
GET request="/T99999999_1.m4v?GOP=30&fps=10&**bitrate=25**&width=176&height=144"

Bandwidth Adaptation by Frames Filtering

Whenever a customer requests streams to the Video Daemon with a specific bitrate and the network cannot respect the bitrate constraint, the Video Daemon has to filter the frames before transmitting. Example: With a 512kbps request with a bandwidth of 128 kbps. The Video Daemon has to cut P frames but also I frames to allow the stream to go through. We can remark that the bandwidth can be managed in the Video Daemon, in the client or somewhere in between.



Bandwidth Management Policy

Presentation

The last SipMultiPlayer video system can display a Multivision on cameras of only one recorder on the same screen. It can also display live and archive video streams. It contains an internal global bandwidth management policy: you can switch from a regulated to a de-regulated monitoring. In regulated mode, SipMultiPlayer makes sure that the total of streams doesn't go past the global bitrate value you have configured. The bitrate value of each video-camera is managed as follows:

- Depends on the number of cameras displayed on the screen.
- The space occupied by each camera on screen: the bigger screen space, the more bitrate is allocated.
- The speed of archives streams (up to 10 times live speed).

Limits of SipMultiPlayer

- No assistance in configuring the system
- No control on the user's configuration
- No real control over the bandwidth if another SipMultiPlayer are connected on the same recorder: requested bandwidth can go past the site's upload capabilities.

Saving Bandwidth Using Video Sensors

Presentation

The recorder system uses MPEG-4 algorithm to produce two permanent video sensors by camera. Those sensors work as follows:

- A rectangle sensor can be drawn on screen
- This picture's area is encoded using the variable bitrate technology
- The bitrate values are stored for 30 days
- Whenever a movement occurs in the area defined by the rectangle, the bitrate value increases.

We can then imagine creating a filter that we put on a video camera stream associated with a threshold. This filter would work as follows:

- While the sensors values are below the defined threshold, the filter cuts the P frames keeping only one I frame every N I frame.
- Whenever one of the sensors goes over the threshold, the filter "releases" the stream letting all P and I frames.

Technically We Provide

- An auto-adjustable threshold to consider the regular image variation during a day (e.g.: light changing).
- Some delay to activate the filter only some seconds when the sensors' value come back below the threshold.

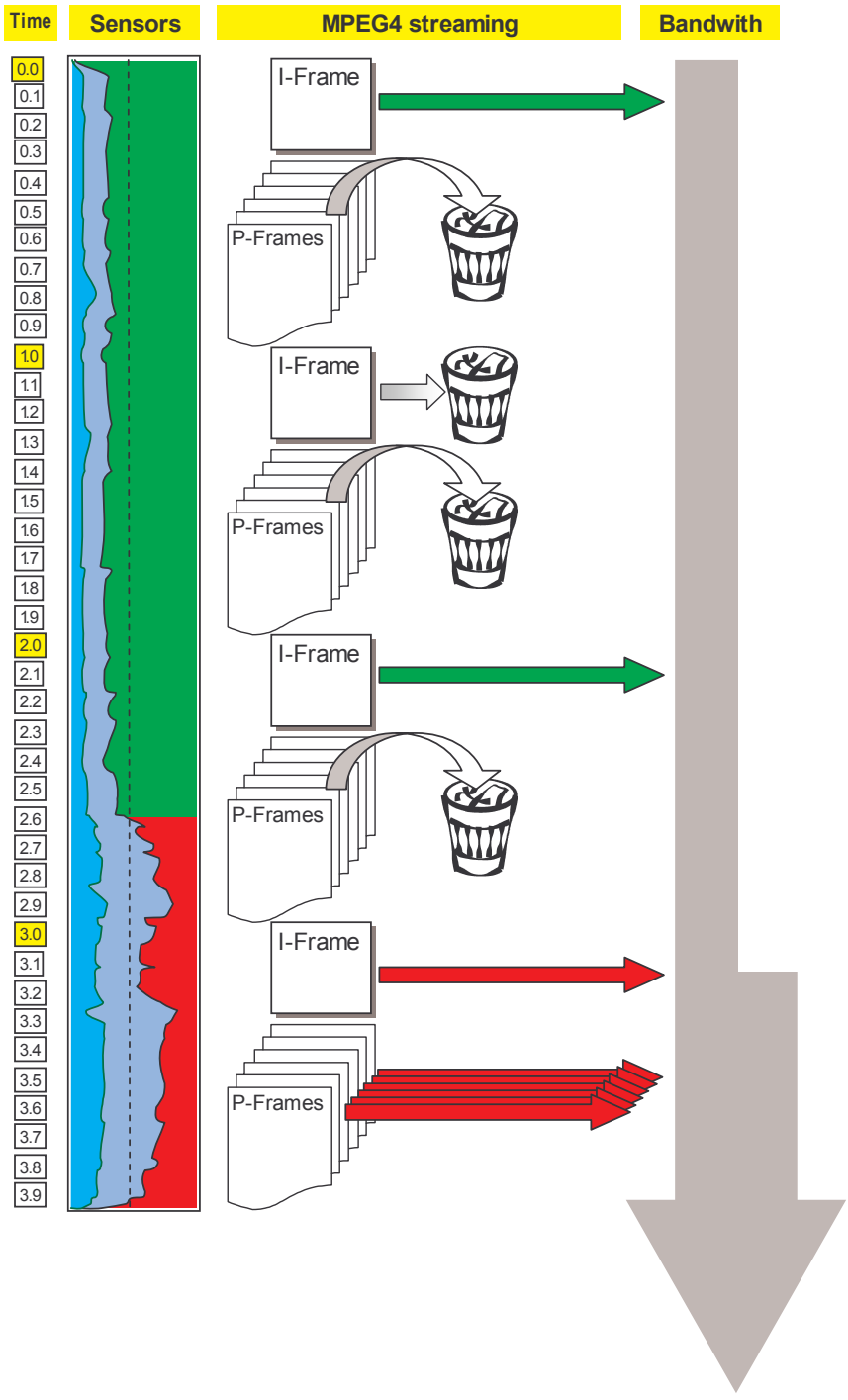
Sensors Implementation

We have to consider the fact that sensors consume processing power in the recorder. This will lower the overall number of FPS that the recorder can produce (say *overall FPS*):

- A sensor taking 25% of the picture's space and encoded at 4 FPS costs 1 overall FPS.
- A sensor taking 25% of the picture's space and encoded at 10 FPS costs 2.5 overall FPS.
- A sensor taking 100% of the picture's space and encoded at 10 FPS costs 10 overall FPS.

We can also think that:

- The more picture space a sensor takes, the more relevant the motion detection will be.
- The higher the sensor's FPS value will be, the less delay the filter will have to release the stream.



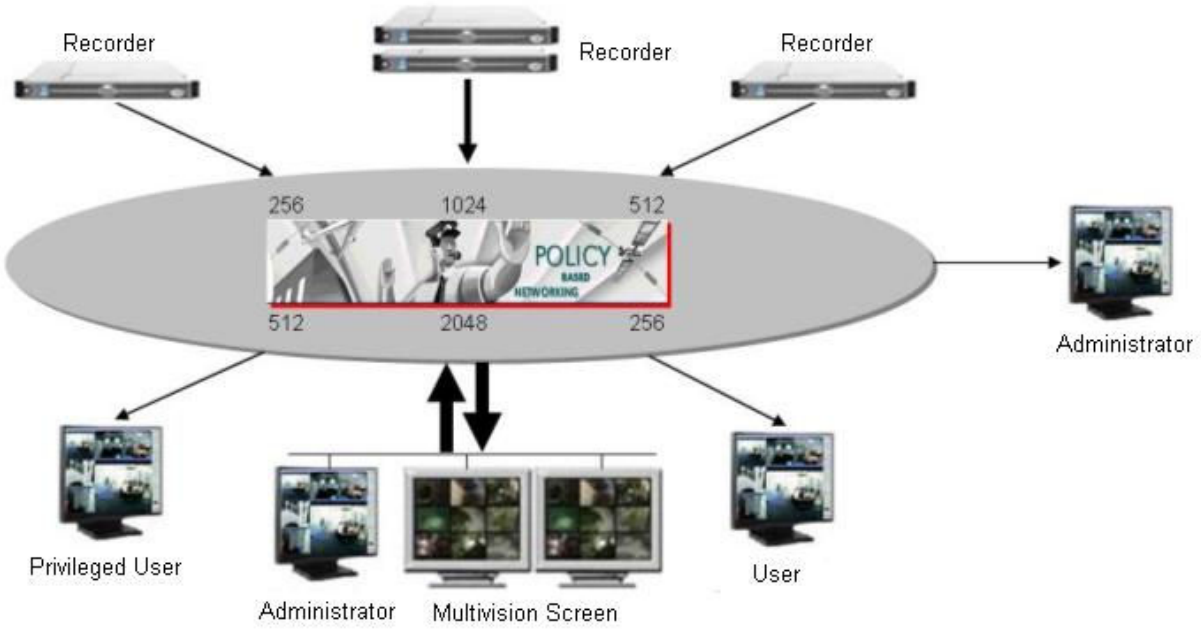
Global Bandwidth Management

The monitored networks' architecture is (generally) as follows:

- Distant sites with one or more recorder with a (different) limited upload for each site.
- A central monitoring (Multivision screen) site with a symmetric access (upload=download)
- User's Multivision screen with limited download
- Administrator access

We can distinguish:

- The administrators who can act without limitations, stop streams, engage filters, etc.
- The privileged user: he is considered in the global bandwidth costs calculations. He can obtain exceptional bandwidth by acting on regular user's bandwidth. Example: The user needs to watch video archives with a maximum comfort. We need to allocate as much bandwidth as possible on the requested cameras.
- The regular user (without any privilege): He is considered in the global bandwidth costs calculation but cannot request for extra-bandwidth allocation.

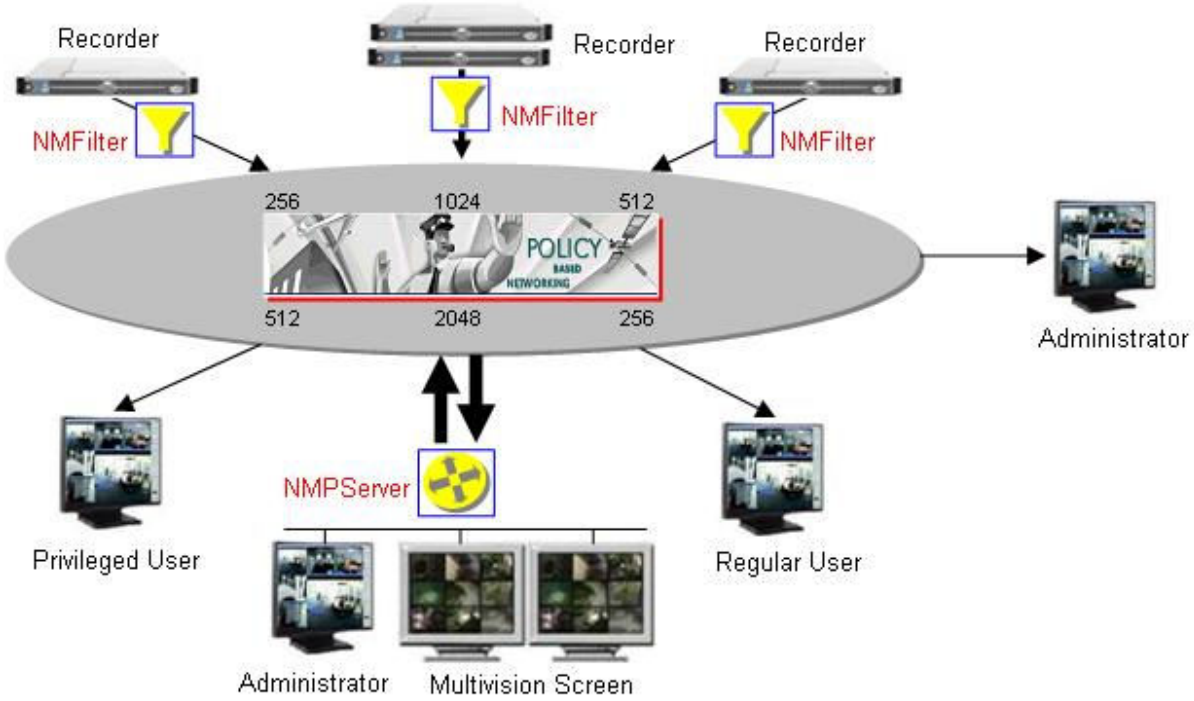


On the picture above we see that the bandwidth management policy must consider:

- The sum of the requested streams for each site and compare it to the site's upload value.
- The sum of the requested streams for the central site and compare it to the central site's download value.
- The sum of the requested streams for every (external) user and compare it to the user's download value.

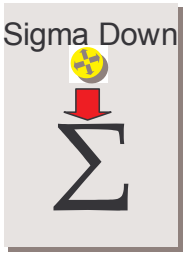
Policy-Based Networking

NMPServer is installed in the central site. It receives all the requests and dispatch all the requested streams. **NMFilter** is installed on the recorders.



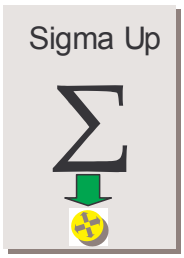
NMPServer is installed in the central site. It receives all the requests and dispatches all the requested streams. **NMFilter** is installed on the recorders. The **NMPServer** and **NMFilter** set have to manage 3 different policies:

- **"Sigma Down" policy**



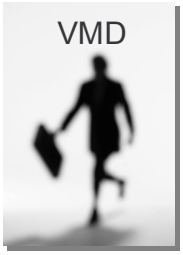
The total bitrate of the streams for a client must not go over the portion of the site's download capabilities that has been allocated for this client.

- **"Sigma Up" policy**



The total bitrate of the streams coming from a recorder must not go over the portion of the recorder's site upload capabilities that has been allocated for this site.

- **"VMD" policy**



VMD, standing for *Video Motion Detection*, describes the mechanisms allowing release of bandwidth for privileged users whenever it is needed.

Technical Issues

Floating Threshold System

I have implemented a floating threshold system to handle the stream release/holding according to the bitrate values. The floating threshold system is composed of three elements:

- A bitrate reader
- A frame reader
- A threshold component

Bitrate Reader

The bitrate reader connects to the video daemon to fetch the bitrate values. It can be called to get the last value read. The values are not stored in any way: the reader is just reading the next value whenever it is told to do so (on-demand reading).

Frame Reader

The frame reader connects to the video daemon to fetch the frames of the live stream. It is only fetching the frames on demand (just like the bitrate reader), but the connection is made at the beginning. The frame reader is not decoding the frames. It is just reading the frames header to return the type of encoded frame (I, P, B).

Threshold Component

The threshold component is a controller synchronizing the bitrate reader and the frame reader in two states: *HIGH* and *LOW*. The *LOW* state (default), says that the component is not activated (threshold value not exceeded). The *HIGH* state says that the component is active (threshold value exceeded). It is acting like a regulator. Indeed, the switching from the *LOW* state to the *HIGH* state is done whenever the threshold has been exceeded. Whereas, coming back to the *LOW* state requires a certain amount of time (passed to the constructor). This avoids going from *LOW* to *HIGH* to *LOW* to *HIGH* ten times in a row.

When created, the threshold component initializes the bitrate and frame reader and it creates a queue of bitrate values. The component reads a bitrate value and a frame. It then decides to keep or drop the bitrate value according to the frame type. Indeed, I-frames always have high bitrate values because they must contain a lot of information. The controller must drop the I-frames bitrate values. When it keeps the value, it is pushed in a queue of limited size. While the queue is not full, nothing is done. But when the queue is full, the next value pushed drops the first value of the queue (FIFO system with limited size). Every time a new value is added to a full-queue, the average bitrate value of the queue is calculated giving a threshold value. The last value pushed is compared to this threshold which determines whether or not the component should switch state.

Whenever the video camera is filming a very active scene, the threshold is almost always *HIGH* so the stream is always released to make sure that the operator (user) can always see the moves.

When the video camera is filming a calm scene, the threshold follows the movement evolution. For instance, when filming a white wall only subjected to the light changes, the threshold quietly adapts to the light change, in the *LOW* state never switching to the *HIGH* state. But, if the video camera is filming a n empty room whose light is switched on or of, the *HIGH* state is reached instantly!

This floating threshold system is a very powerful one even though it is based on a fairly simple algorithm. The main difficulty is to synchronize the bitrate value and the frame value to make sure that we don't drop the wrong bitrate values. This issue has been addressed in the previous section.

Frames Filtering Problem: Bitrate Reading Delay

I managed to implement a frame filter decoding the frame type on the fly. That is, I have created a frame reader which, without decompressing the frames, read the MPEG-4 header to determine the frame type (I, P, B) directly in the stream. The mechanism decides to drop or keep them according to the policy which has been set. This works fine on its own. Then, I had to create a bitrate reading system. The bitrate reader connects to the video daemon requesting the bitrate encoded on the live stream. The bitrate stream is just composed by integers separated by end of line symbols (CRLF). The bitrate reader was working fine on his own also (see *Floating Average Threshold System*). The problem arose when trying to synchronize the bitrate reader with the frames reader. Indeed, it appeared that the bitrate values returned by the bitrate reader did not correspond to the values of the frames which were "supposed" to be generated at the same time. Indeed, after code auditing, I noticed that it could be possible for the frame reader to be delayed because of network problems. The bitrate values were no more corresponding to the frames, so the filtering could not be accurate.

Given these information, the system had to be re-designed. In fact, the NMFilter component cannot be separated from the video daemon. The video daemon must manage the filtering directly in the frame encoding thread so that it can make the frame and the bitrate values match directly. These two entities are impossible to separate unless you put some kind of time markers in both the frames and the bitrate values to make them match. The fact that the video daemon is the very complex system prevented me to go further alone on this way.