MEDIA SERVER FOR VIDEO AND AUDIO EXCHANGE BETWEEN THE U-70 ACCELERATOR COMPLEX CONTROL ROOMS

I.V. Lobov, V.G. Gotman, IHEP, Protvino, Russia

Abstract

The media server was developed that implements the exchange of video and audio streams between control rooms for U-70 technological subsystems. Media server has the possibility of making changes into the intermediate video images to embed current technological information. The media server is implemented as a set of threads of execution, one for each video format conversion module. The media server is a chain of successive transformations of video and audio streams from one format to another: H.264—Y4M—THEORA formats for video, PCM—VORBIS formats for audio. The final video and audio streams are encapsulated into the OGG container stream which is translating into the local network. OGG container has been chosen because of its completely open, patent-free technology and full support in HTML5. Any Web browser with full HTML5 support may be used as OGG stream consumer. The browser client program has written with tag <video> utilization. This allows for client to work on different platforms (Linux, Windows) and get rid of third-party video plug-ins.

INTRODUCTION

The aim of the work was to develop a dispatching system for the organization of audio and video-sharing between different U-70 technological subsystems in IHEP. Requirements for the dispatching system were as follows:

- Simple and convenient instrument of organizing the conversations and conferences.
- The client software must run on different operation systems.
- The software must use open-source free algorithms and libraries.
- Do not use any special designed programs (plug-ins) on the client side.
- The ability to modify the intermediate video images in real time scale.
- To record the video and audio tracks into archive with a possibility of quick search of the desired fragments.
- The ability to transmit media information in conjunction with digital technology data.

DISPATCHING SYSTEM STRUCTURE

The solution of the task lies in the following main ideas (see Fig. 1):

- The dispatching system will use the IP-cameras with video and audio transmit ability instead of connecting to the PC webcams.
- The media server will provide data transmission from IP-cameras to the clients in form of media streams. Thus, instead of a set of programs for different client's operation systems only one dispatching program will be written in.
- To use the benefits of the HTML5 for the client access to the server.

IP-camera which makes your participation in the conference does not need to be connected to the client computer. Thus the client computer does not transmit any media-streams to the server. It simply receives the media-streams. At first, the client makes a connection to the server. Next, the server connects to an IP-camera and starts to receive the media-stream. Finally, the client begins to receive the media-stream from the server.

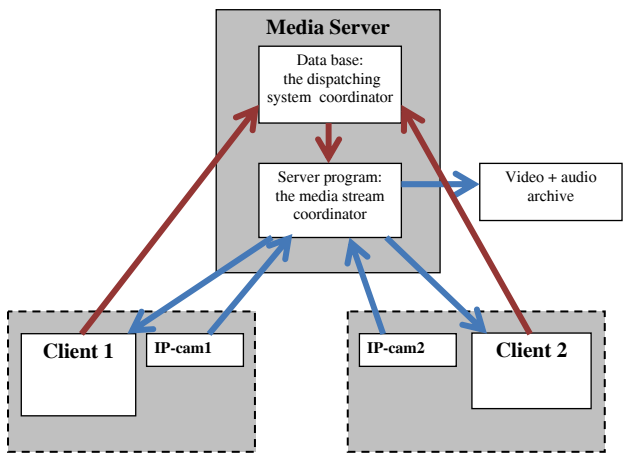

Figure 1: The scheme of the data flow (blue arrows) and the control flow (red arrows).

SOFTWARE INSTRUMENTS AND MEDIA FORMATS STANDARDS

Programming Tools

The server part of the dispatching system was written in Visual Studio 2012.

The libraries used in the project:

- JM 18.6,H.264/AVC Software, Karsten Suehring [1].
- libogg, version 1.3.2, Xiph.Org [2,3].
- libtheora, version 1.1.1, Xiph.Org [4].
- libvorbis, version 1.40, Xiph.Org [5].

The client part was written in HTML5 + JavaScript.

ISBN 978-3-95450-170-0

Media Exchange Standards

RTSP—Real Time Streaming Protocol designed to control streaming media servers [6].
RTP over H.264—Real time Transport Protocol designed to transmit H.264 video streams in real time scale [7].

Media Data Formats and Standards

H.264, MPEG-4 Part 10 or AVC (Advanced Video Coding)—licensed video compression standards designed to achieve a high compression ratio while maintaining high video quality [1].

YUV—image color model, in which the color appears as three components: the luminance (Y) and two chrominance components (U and V). Luminance component Y contains a "black and white" grayscale image, and the remaining two components contain the information required to restore the color. YUV is convenient model for image recognition and intermediate image replacement by changing the desired pixels. YUV model will help in the future augmented reality implementation which is intended to use in the dispatching system [1].

OGG—open standard multimedia container format, it is the main file and stream format for multimedia codecs funded by Xiph.Org [2].

THEORA—free video codec for video compression with losses developed by the Xiph.Org [4].

VORBIS—free audio compression format with losses [5].

The OGG format with VORBIS and THEORA content is quite promising because of 1) free licensing and 2) the ratio between sound and image quality and file size which is the best among peers. The last factor is quite critical for local networks when a large amount of packages is used for media exchange. Owing to these, the OGG format was selected as a container for the audio and video transmission to the client.

SERVER PROGRAM DESCRIPTION

The server software is designed for transmitting data streams between clients as shown in Fig. 2.

The transmission of the media from IP-camera to a web browser is a chain of successive media transformations from one format to another. The chain for media transformation from IP-camera to the client is performed by the server program. The server program connects to an IP-camera, takes video and audio RTP-packets from the camera. Then the received media information is analyzed and altered (if necessary). Finally the media information is sent to the client.

The server program consists of the software modules which transform the information as follows:

- Module "MAIN"—establishes a connection to IP-camera via RTSP and receives the data in real time via RTP.
- Module "YUV"—converts the compressed video format H.264 into uncompressed image format YUV for further analysis.
- Module "OGG"—performs the transcoding of the pair "Uncompressed video format YUV + Audio format PCM [8]" into the THEORA and VORBIS formats with subsequent encapsulation into an OGG container.
- Module "TRAN"—transmits the OGG packets to the client. The module connects to the database and reports on its readiness for data transmission.

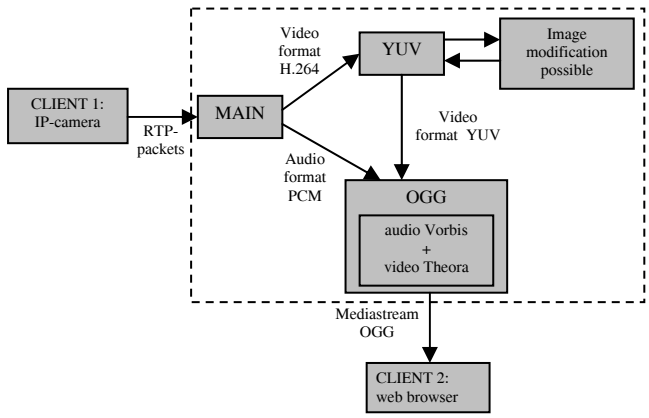

Figure 2: The data flow.

Program Organization as a Set of Threads

Program modules are implemented as separate threads, one thread for each client. The data exchange between threads is performed by means of intermediate buffers and semaphores in order to synchronize the intermediate buffer's reading and writing. The semaphores are needed to prevent the possibility of data corruption when threads use the intermediate buffers.

Four threads are used to transfer video and audio data from a camera to a client. Thus, the data exchange between two clients requires eight threads. The same software procedures are used for both clients. This imposes a requirement of using only re-entrant software. In other words, it is necessary to use only two types of variables:

- Local variables in procedures;
- In case of using global variables they should be organized in the form of two-dimensional arrays.

Description of the MAIN Module

The main module of the server is started as a thread for each client. It connects to the camera via RTSP. MAIN module generates three threads (YUV, OGG, TRAN) for processing and transmitting information to the client browser. The task of MAIN is to take RTP packets from the camera, analyze them and identify the types of the packets (audio or video, single or fragmented).

Each RTP packet contains the NAL units of different types [7]:

- SPS—the Sequence Parameter Set.
- PPS—the Picture Parameter Set.
- IDR—I frame.
- NONE-IDR—B- frame (move).

ISBN 978-3-95450-170-0

The flow of the MAIN is implemented in infinite loop which accepts incoming packets from the camera.

For video data, MAIN checks the packet loss, and if the package was lost, the buffer is filled with the last full frame received. In case no packet loss the buffer is filled with all of the packets had come between two consecutive packets containing SPS. Once the buffer is full, the flow is stopped and the video data is copied to the YUV module. When the copy is finished the flow continues.

In the contrary, audio data is sent directly to the OGG module.

Description of the YUV Module

This software module is implemented in the form of an infinite loop waiting for data from the main program module (MAIN). After receiving the data, the YUV decodes them from a compressed format H.264 to uncompressed YUV format and transmits the transcoded data to the OGG module.

Description of the OGG Module

The first step of the OGG module is to initialize OGG header with THEORA and VORBIS content. OGG module writes the header to the output buffer and begins to wait for data from the YUV module, as well as the TRAN module readiness. Upon the receiving the uncompressed video data from YUV module, the OGG module breaks them into frames. One uncompressed frame makes IDR-frame while the others make NONE-IDR frames (move). At the same time the OGG module waits for the uncompressed PCM audio data from the MAIN and encodes it in a compressed format VORBIS.

Finally, when the THEORA and VORBIS formats are encoded into OGG container packets, the output is written to the intermediate buffer for the TRAN module processing.

Description of the TRAN Module

The TRAN module is waiting a request from the client to join. After joining the TRAN module gets the data from intermediate OGG buffer, divide it into packets with 1400 bytes length and sends them to the client.

CLIENT PROGRAM DESCRIPTION

The client program is implemented in the form of a window containing the video and control buttons as shown in Fig. 3. It connects to the database and sets the "online" flag for other clients. At the same time it reads the "online" flags from all other clients and the server.

The operator can connect to other clients and establish connection with them. He can press the corresponding button addressed to the desired remote client. After getting answer the remote connection is established and the video picture comes alive.

The connection buttons use 3 colors:

- Green—the client (server) is online.
- Red—the client (server) is offline.
- Yellow—operator requests the connection.

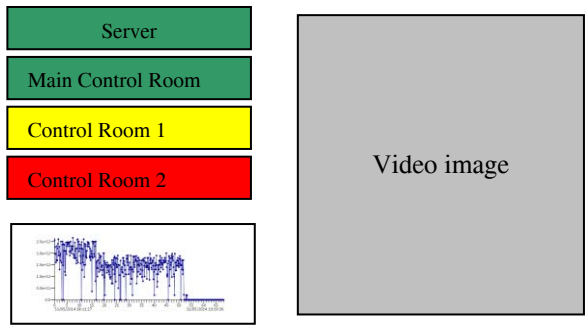

Figure 3: The scheme of the client interface.

CONCLUSIONS

Upon the development process of the media server software a number of technical problems was resolved:

- Initially the chain of video stream consecutive format conversions resulted in data loss. The solution founded—a media server was organized as a set of multiple interacting threads of execution.
- Initially the audio and video streams were gradually diverging from each other due to the differences in timing principles (audio is transmitting in frequency units while video is transmitting in video frames). The peculiar method of the audio and video stream synchronization was developed—to use the independent timing scheme, same for audio and video.
- Initially the image was transferred choppy due to the inconstant number of frames per second getting from the camera. The problem was solved by counting the actual number of frames per second.

A dispatching system scheme for U-70 Accelerator Complex was developed and implemented. An optimal method for the video and audio streams conversion was achieved via threads of execution. As a consequence the video and sound streams have no transcoding losses whereby the media is played smoothly and correctly.

REFERENCES

[1] Karsten Suehring et al., H.264/AVC Software Coordination; http://iphome.hhi.de/suehring/tml/
[2] S. Pfeiffer., The Ogg Encapsulation Format Version 0, RFC 3533; http://xiph.org/ogg/doc/rfc3533.txt
[3] I. Goncalves et al., Ogg Media Types, RFC 5334; http://xiph.org/ogg/doc/rfc5334.txt
[4] Xiph.Org Foundation, Theora Specification; http://theora.org/doc/Theora.pdf
[5] Xiph.Org Foundation, Vorbis I specification; http://xiph.org/vorbis/doc/Vorbis_I_spec.pdf
[6] R. Lanphier et al., Real Time Streaming Protocol, RFC 2326; http://www.ietf.org/rfc/rfc2326.txt
[7] Y.-K. Wang et al., RTP Payload Format for H.264 Video, RFC 6184; http://tools.ietf.org/html/rfc6184
[8] J. Salsman et al., The Audio/L16 MIME content type, RFC 2586; http://tools.ietf.org/html/rfc2586

ISBN 978-3-95450-170-0