A Method of Pseudo-Live Streaming for an IP Camera System with HTML5 Protocol

[1]Paul Vincent S. Contreras, [2]Jong Hun Kim, [3]Byoung Wook Choi
[1]*Seoul National University of Science and Technology, Korea, vinz.contreras@gmail.com*
[2]*Seyeon Tech, Co., Ltd., Korea, kjh@seyeon.co.kr*
[*3]*Seoul National University of Science and Technology, Korea, bwchoi@seoultech.ac.kr*

Abstract

This paper presents a way to provide pseudo-live streaming video from an Internet Protocol (IP) camera system to client computers using an internet browser that can display Hyper Text Markup Language 5 (HTML 5) pages. This is called pseudo-live streaming because the video stream is not packed in streaming protocols, such as Real-time Transport Protocol (RTP). This method does not involve the installation of plugins in the client side as most of the modifications are done on the server side, which in this case is the IP camera. The initial step in providing the mentioned functionality is presented here. The initial research involved the development of a tool to convert packetized elementary streams that contains extra manufacturer data into an elementary stream. The developed tool was able to convert packetized elementary streams from an IP camera into an elementary stream. The developed tool can be used to as a building block to provide a pseudo-live streaming experience.

Keywords*: HTML5, IP Camera, MP4, Pseudo-live streaming, Streaming*

1. Introduction

IP camera systems are one of the most sought items of many businessmen in order to keep their businesses secure. Unlike traditional Closed Circuit TV (CCTV) cameras, IP cameras allow its owners to view live streaming videos wherever there is an internet connection. In order to show users a live video stream from their IP cameras, IP cameras broadcasts video streams via streaming protocols. Users can view these video streams using a player of their choice.

To provide convenience, IP camera manufacturers let its clients view video streams from their IP cameras through web browsers. A user would be able to view live video streams using their web browsers provided that they run special plugins developed by the IP camera manufacturers. These plugins allow the user to play objects embedded in HTML pages. For IP camera users, these plugins should be installed first in their computers before any kind of live video stream will be made viewable. However, the idea of convenience is lost when an IP camera owner wishes to view video streams from a computer which he does not own. Typically, these computers don't provide elevated privileges to allow the installation of plugins made by the IP camera manufacturers.

In the past decade, improvements to HTML have been introduced. HTML 5 was introduced as an improvement of the current HTML 4.01 standard. HTML 5 introduces new media elements, `audio` and `video` [1]. These elements can be used to play audio and video in a web page without running any kind plugin to handle streaming audio and video.

One of the many file formats that many HTML5-compatible browsers support is the MP4 container format with videos coded using H.264 [2-5]. Many IP camera systems use H.264 to code their videos. These H.264-coded videos are usually broadcasted for use in streaming protocols, such as RTP and RTSP.

Unfortunately, the HTML 5 specifications for the video element do not support streaming from live video sources, such as RTP and RTSP. HTML 5 browsers currently support progressive streaming over the HyperText Transfer Protocol (HTTP) [6-8]. This poses a problem to web developers and IP camera system developers who want to show live streaming videos without installing any plugin on the client's computer because streaming media protocols are not supported.

In this paper, a method to would provide live video stream from an IP camera by exploiting two of the characteristics of the MP4 file format is presented. This idea is based from the idea that an MP4 file

* Corresponding author

the ability to be written by an encoder and then read by a decoder simultaneously. The end result is a progressive stream of video from an IP camera system.

There are two phases in this research. The first phase involves the development of a tool at the server side. The second phase involves the porting, testing and deployment of the developed tool into an actual IP camera. In this paper, the first phase is presented.

2. Video streams and HTML 5

2.1 Video Streams

Video streams are usually encapsulated into packets to allow transmission in a computer network. Encapsulation is the process of taking a video stream, formatting it into IP packets, and adding headers and other data in order to comply with a specific protocol. Encapsulation is a required step before a video stream is transported into an IP network.

There are many kinds of video streams MPEG has defined. First is the elementary stream. An elementary stream is the raw output from a video encoder. A video encoder will continue to create an elementary as long as it is activated. Also, an elementary stream is the standard input for a video decoder. This stream, in nature, is continuous. The second kind of stream is the packetized elementary stream. This stream is an easier-to-handle version of an elementary stream. This stream is just an elementary stream broken down into chunks. These chunks are often called packets (but not similar with IP packets). These packets usually contain headers. The header can contain timing information that allows audio and video to be synchronized. Third is the program stream. Program streams carry a single program. A program is a combination of video, audio and other related data makes a complete audiovisual experience. Last is the transport stream. A transport stream is similar to a program stream but is specifically designed for transmission along an unreliable network. A transport stream usually has a fixed size of 188 bytes.

Because of its size, an elementary stream can't be transported along a computer network. It is usually converted to other kinds of streams before transmission. After conversion, they are encapsulated into IP packets.

Videos stored locally in a computer contain elementary streams of videos. Since elementary streams only contain one particular type of data, a container format is used to encapsulate both audio and video [9-10].

2.2 HTML 5

HTML 5 is the revision of the current HTML standard, which HTML 4.01. HTML 4.01 was last published last December 2000. HTML 5 was seen as an improvement of HTML 4.01 and is also designed to replace HTML 4.01 and XHTML. HTML 5 adds more features to HTML 4.01 and eliminated features in HTML 4.01 that were never used. The HTML 5 specification is currently not yet final and is currently a Candidate Recommendation by the World Wide Web Consortium [11].

Unlike HTML 4.01, HTML 5 code is more readable and cleaner. HTML 5 also adds multimedia features. Previously in HTML 4.01, developers need to add generic objects in web pages. Some of these objects require the use of plugins in order to be displayed correctly. For example, in order to play an MP3 file, either a Windows Media Player Active X plugin or a Flash plugin is required. HTML 5 makes it also easier for developers who use JavaScript because an API is provided for cooperating with JavaScript code.

One of the features that are of particular emphasis is the `video` element of HTML 5. It allows playback of videos without using any kind of plugin. However, this element of HTML 5 only allows the progressive streaming of videos, such as the mechanism done by YouTube [12]. In progressive streaming, a video can is downloaded and then played simultaneously unlike real streaming, wherein after a video frame is played, it is discarded [9].

Web browsers that display the `video` element currently support the playback of videos packed in MP4 file containers that use H.264 and WebM. However, there is no consensus up to now whether which of the two video containers will be used. Previously, MP4 files packed in H.264 was suggested as the primary format of videos in the <video> element. For now, whichever format to support lies on the web browser vendors [13].

2.3 Progressive Streaming of MP4 Files

In order to display an elementary stream in a web browser that supports HTML 5, the stream should be first encapsulated in a video container. The stream should either be encapsulated in an MP4 file container or the WebM container.

The MP4 File Format, or officially the MPEG-4 Part 14, is a file format developed the Moving Pictures Experts Group (MPEG) of the International Organization for Standardization (ISO) [14]. On the other hand, WebM is a video container whose development was sponsored by Google, Inc. The MP4 File Format currently supports the encapsulation of H.264 video, which most IP camera manufacturers use [14]. On the other hand, WebM only supports the video codecs developed by Google, which is VP8 and VP9 [12]. Both formats allow streaming of videos.

As an extension of the ISO Base Media format, an MP4 files also contain boxes. The MP4 file format contains boxes that hold information. One of particular emphasis is the `moov` box. This box acts as the table of contents of an MP4 file. Some of the things that the `moov` box describes are the size of the video and the kind of codec the video uses [15].

The placement of the `moov` box inside in an MP4 file is of great importance. Web browsers that support HTML 5 pages can only start playing MP4 files as soon as they can read the `moov` box. The `moov` box is usually placed by video encoders at the end of the file. To enable playback even though an MP4 file is not yet finished downloading, the `moov` box should be placed near the start of the MP4 file. If the `moov` box is placed near the end of the file, the entire video should be downloaded first in the client computer before it can be played [16].

3. Environment Setup

In this research, Seyeon Tech FW1173-MS IP camera was used [17]. The said IP camera has the following specifications:

- 1/2.9” 2.43M PS CMOS Sensor
- DC Vari-focal auto iris Lens
- Min. Illumination: 0.0016Lux, TDN
- Max. 30fps/Full HD(1920x1080)
- Dual Stream(H.264, M-JPEG)
- Micro SD slot, USB 2.0
- Option : POE, Lens

Video feeds from the IP camera can be accessed using a web browser. This IP camera can broadcast high quality videos at 30 frames per second at 1920x1080 in progressive mode. This IP camera can broadcast videos coded either in H.264 or Motion JPEG (M-JPEG). The IP camera can also broadcast with bitrates from 32 kilobits per second to 12 megabits per second.

During the experiment, the following were the settings during of the IP camera:

- Video codec was set to H.264
- Frame rate was set at 30 fps
- Video size was set at 704 x 480 pixels
- Bit rate was set to unrestricted

The set-up of the first phase of this study is depicted in the diagram below:

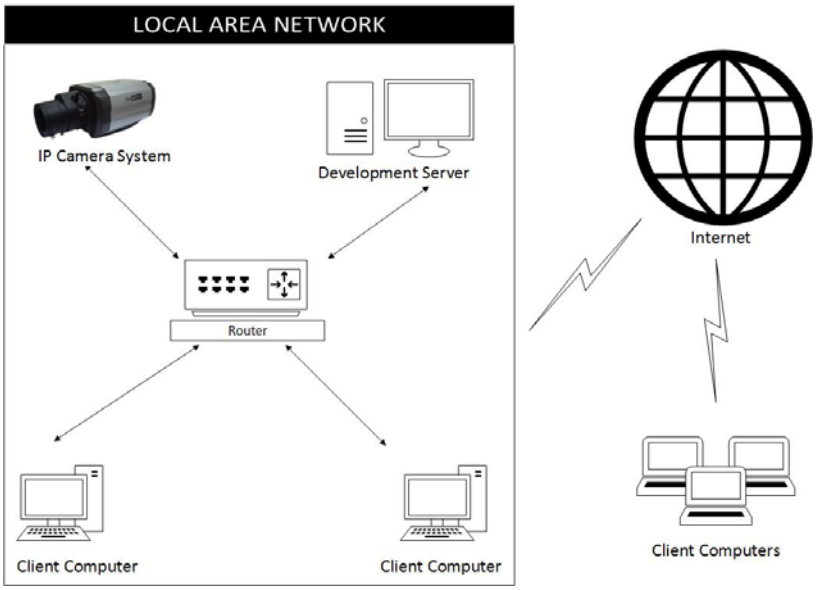

Figure 1. Phase 1 development environment

As seen from Figure 1, the IP camera is connected to a router. The development server is also connected to the router. Both the IP camera and the development server are accessible in the internet because the router has an assigned static IP address. The IP camera and the development server were assigned different ports. Client computers can either belong to the local area network or outside the local area network.

The way to get packets from the IP camera is through CGI commands. These CGI commands are typically executed in a web browser. The connection between the IP camera and the server is a HyperText Transfer Protocol (HTTP) connection. Therefore, there is no data loss between the IP camera and the development server. These packets contain H.264 frames, which can either be I-frames or P-frames. Unfortunately, the user cannot specify which frame to get, whether I-frame or P-frame.

4. Methodology

IP cameras can broadcast video frames either as packetized elementary streams or as transport streams. But since browsers that support HTML 5 only allow progressive streaming of videos, web browsers cannot process any other kind of streams except elementary streams. Even though the IP camera has support for streaming protocols, such as Real Time Protocol (RTP) and Real Time Streaming Protocol (RTSP), web browsers cannot play a video stream through the said protocols.

To be able to stream from an HTML 5 web browser that uses the `<video>` tag, streams from the IP camera need to be converted first back to elementary streams. Since all HTML 5 web browsers don't play raw H.264 elementary streams and since browsers only support playback of videos in a video container file, the only way to do this is to encapsulate the elementary stream data into the video container format. Since most IP cameras use the H.264 as a way of compressing videos and since most web browsers support H.264, the MP4 file format is used.

An overview of the methodology is shown in Figure 2 below:

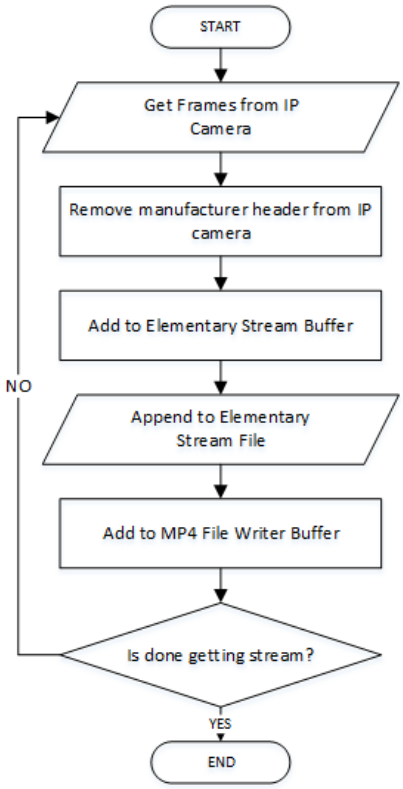

Figure 2. Phase 1 methodology

The experiment involves the development of a C++ program that executes the CGI command that fetches IP camera packets. The program removes the header of these packets because these headers contain extra information that is not needed by an H.264 elementary stream decoder. The raw packetized elementary stream is added to an MP4 file writer buffer and then writes it to an MP4 file. The structure of a typical IP camera packet from the Seyeon Tech IP camera is depicted in the following diagram:

Figure 3. IP camera frame structure

As seen from Figure 3, the frame contains manufacturer information and the payload. The payload contains the raw packetized elementary stream frame. This frame can either be an H.264 I-frame or P-frame. This payload is an H.264 Network Abstraction Layer (NAL) Packet.

Elementary streams are organized as a series of synchronization markers and the NAL unit (refer to Figure 4). On the other hand, a packetized elementary stream only contains one synchronization marker and one NAL unit.

Figure 4. An elementary stream

The start of the raw packetized elementary stream is usually started with a series of numbers called synchronization markers. These synchronization markers can have the value 0x0000001, 0x0000002 or 0x0000003 [18].

5. Experimental Results

Frames from the IP camera were fetched and then these frames were processed in order to extract the raw H.264 packetized elementary stream. These raw packetized elementary streams were added to a buffer and then were written to separate files.

First, packets were fetched and no modification was done to the frames. Each individual packet was saved to separate files. On the second run, a new set of packets were fetched from the IP camera and the developed tool removed the IP camera manufacturer header from the packet. These packets were also saved to separate files. On the third run, packets were fetched and these packets were modified. These packets were saved and appended to one output file.

The IP camera always returns 1 I-frame followed by 15 P-frames. The structure of the first I-frame and first P-frame were examined.

Figure 5 shows the hexadecimal view of the first bytes of an I-frame during the first run. Highlighted are bytes with addresses 0x0000h to 0x0043h. These bytes contain the IP camera manufacturer data. Seen after the highlighted addresses is the synchronization marker. The synchronization marker in Figure 5 contains the data, 00 00 00 01 from address 0x0044h to 0047h respectively.

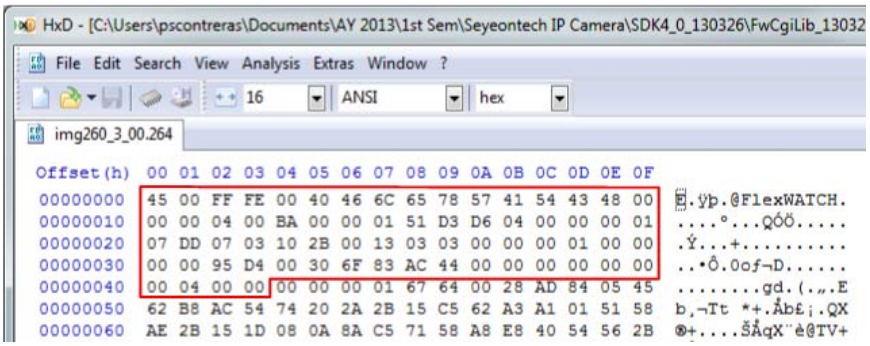

Figure 5. Contents of a sample IP camera packet, I-frame with manufacturer header

Figure 6 shows the hexadecimal view of the first bytes of an I-frame after the header data was removed from the packet. Comparing with Figure 5, the data from 0x0044h were moved to the start of the file. The synchronization marker was moved to the start of the file.

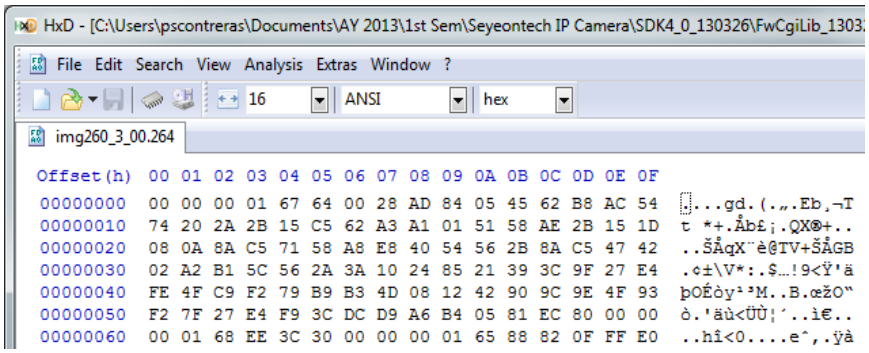

Figure 6. Contents of a sample IP camera packet, I-frame, manufacturer header removed

Similarly, the following shows the hexadecimal view of the first bytes of a P-frame. Bytes in address 0x0000h to 0x0043h contain the camera manufacturer header information. Bytes 0x0044h to 0x0047h show the synchronization marker.

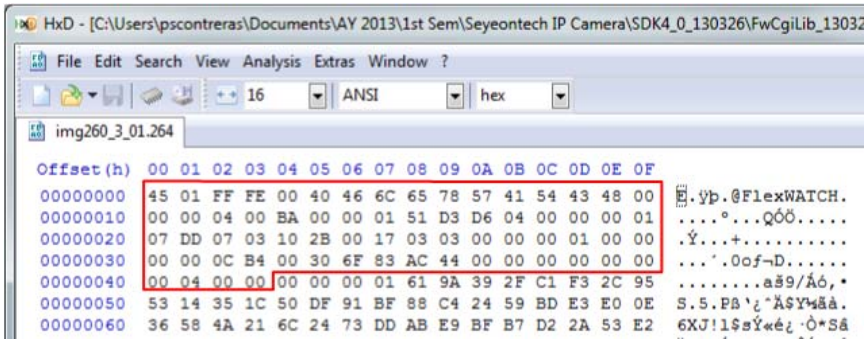

Figure 7. Contents of a sample IP camera packet, P-frame with manufacturer header

Figure 8 shows the hexadecimal view of the IP camera packet after the removal of the camera manufacturer header. The synchronization marker is moved to the start of the file.

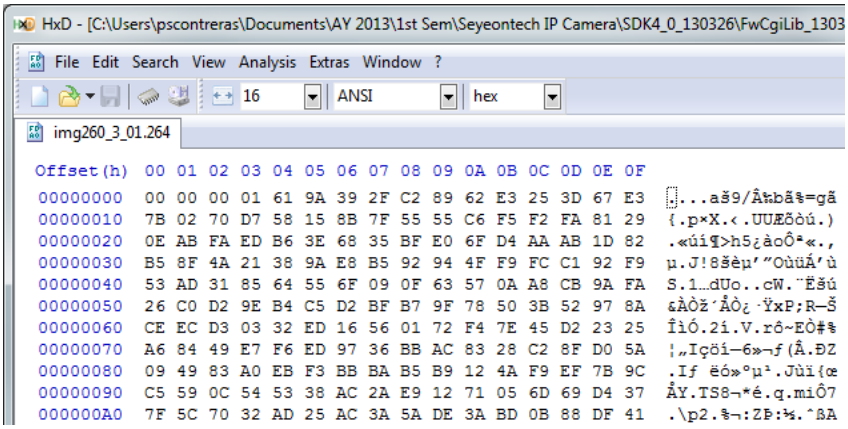

Figure 8. Contents of a sample IP camera packet, P-frame, manufacturer header removed

To determine whether the experiment was a success, another trial was run, but instead of putting each frame into a separate file, it was combined into one file. The scene recorded using the video camera involved the adjusting the position of the IP camera's position.

The output file was then played in a player that can play raw H.264 files. VLC Player was used to check whether the output file was indeed playing. Below are screen shots of the output file of this experiment.

6. Conclusions

This paper presented the initial steps of providing a mechanism to implement live streaming without the use of plugins. However, this research only showed of converting packetized elementary streams from an IP camera to elementary streams. Research and development of integrating the resulting elementary stream into the MP4 file container should be done. It is also not known whether an MP4 file can have infinite time duration, so more research should be done to determine the capability of the MP4 file format. Once this has been done, the developed tool can then be recoded again as a Linux FUSE program so that a file will appear like a normal file in the file system, but in reality it is a program that runs and outputs a video stream from the IP camera.

Finally, the developed mechanism should be ported into IP camera which is an embedded system and play an embedded web server to do broadcasting.

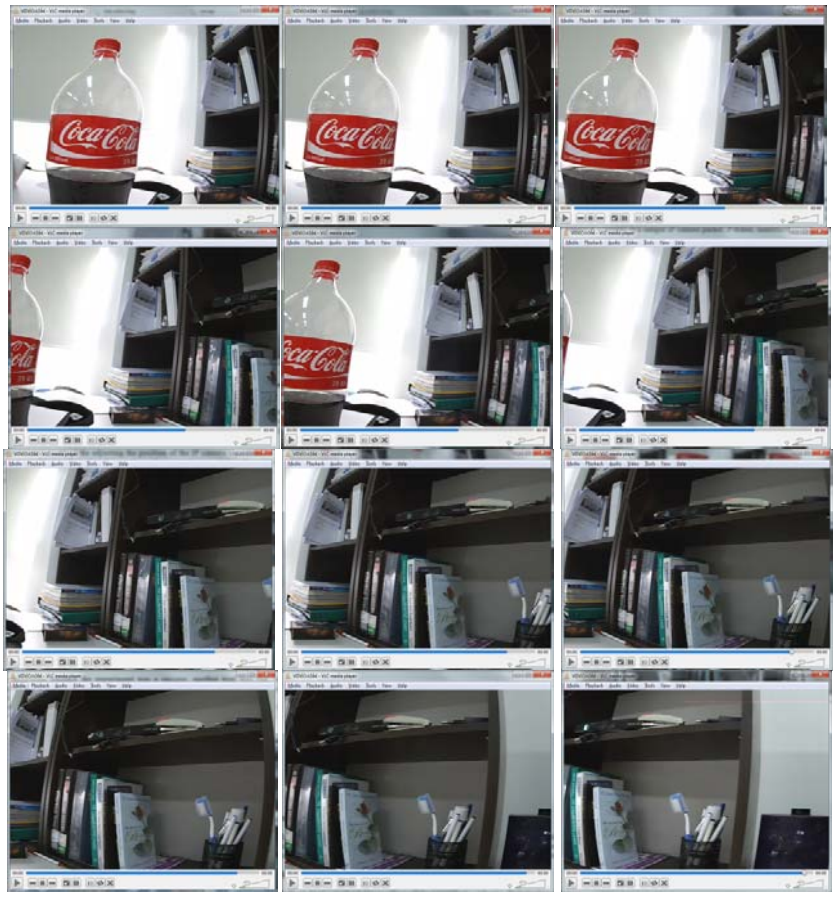

Figure 9. VLC Player playing the output video file

Acknowledgments. This study was financially supported by Seoul National University of Science and Technology.

7. References

[1] W3 Schools, “HTML5 New Elements”, http://www.w3schools.com/html/html5_new_elements.asp.
[2] LongTail Video, “The State of HTML5 Video”, http://www.longtailvideo.com/html5.
[3] Soon-kak Kwon, A. Tamhankar, K.R. Rao, “Overview of H.264/MPEG-4 part 10”, Journal of Visual Communication and Image Representation, vol. 17, issue 2, pp. 186-216, 2006.
[4] Seung-Hwan Kim, Jin Heo, Yo-Sung Ho, “Efficient entropy coding scheme for H.264/AVC lossless video coding”, Signal Processing: Image Communication, vol. 25, issue 9, pp. 687-696, 2010.
[5] Ashish K. Banerji, Kannan Panchapakesan, Kumar Swaminathan, “Stitching of H.264 video streams for continuous presence multipoint videoconferencing”, Journal of Visual Communication and Image Representation, vol. 17, issue 2, pp. 490-508, 2006.
[6] “HTML5 Video Video RTSP Live Video Streams”, http://www.jquery4u.com/flowplayer/html5-video-rtsp-live-streams.
[7] Jiwoong Bang, Daewon Kim, “Efficient RTSP-based multiple buffering and packet transmission methods for delivering OMA PoC Box service”, Computer Networks, vol. 56, no. 15, pp. 3468-3478, 2012.

[8] Robert Kuschnig, Ingo Kofler, Michael Ransburg, Hermann Hellwagner, “Design options and comparison of in-network H.264/SVC adaptation”, Journal of Visual Communication and Image Representation, vol. 19, no. 8, pp. 529-542, 2008.
[9] Simpson, Wes., Video Over IP: IPTV, Internet Video, H.264, P2P, Web TV, and Streaming: A Complete Guide to Understanding the Technology. Elsevier Inc, 2008.
[10] C Herpel, A Eleftheriadis, “MPEG-4 Systems: Elementary stream management “, Signal Processing: Image Communication, vol. 15, issues 4–5, pp. 299-320, 2000.
[11] W3 Working Draft, “Differences from HTML 4”, http://www.w3.org/TR/html5-diff/
[12] The WebM Project, “About WebM”, http://www.webmproject.org/about/
[13] “HTML5 Tracker”, http://html5.org/tools/web-apps-tracker?from=1142&to=1143
[14] International Organization for Standardization, Information technology – Coding of audio-visual objects – Part 14: MP4 file format. International Organization for Standardization, 2008.
[15] International Organization for Standardization, Information technology – Coding of audio-visual objects – Part 12: ISO base media file format. International Organization for Standardization, 2008.
[16] Levkov, Maxim. “Understanding the MPEG-4 movie atom”, Adobe Corporation, http://www.adobe.com/devnet/video/articles/mp4_movie_atom.html
[17] Seyeon Tech Co., Ltd., “Specifications of Seyeon Tech FW1173-FX”, http://www.flexwatch.com/products/ipcamera/box/1173_fx.asp
[18] Gentle Logic, “Exploring H.264. Part 2: H.264 Bitstream Format”, http://gentlelogic.blogspot.com/2011/11/exploring-h264-part-2-h264-bitstream.html