International Journal of Software Engineering and Its Applications
Vol.8, No.6 (2014), pp.39-50
http://dx.doi.org/10.14257/ijseia.2014.8.6.04

Implementation of HTTP Live Streaming for an IP Camera using an Open Source Multimedia Converter

Gil Jin Yang[1], Byoung Wook Choi*[2] and Jong Hun Kim[3]

[1, 2] Dept. of Electrical and Information Engineering, Seoul National Univ. of Science and Technology, Seoul, Korea,
[3] R&D Center, Seyeon Tech Co., Ltd., Seoul, Korea
gjyang@seoultech.ac.kr, bwchoi@seoultech.ac.kr and jhk@seyeon.co.kr

Abstract

Hyper Text Markup Language (HTML) is the main markup language for creating web pages and other information that can be displayed in a web browser. HTML5 is the fifth revision of the HTML standard. Its aim is to deliver almost everything you want to do without requiring additional plug-in. HTML5 first describes new tags for multimedia, which are the new audio and video tags. We produced an implementation of HTTP Live Streaming (HLS) for an IP camera by using elements of HTML5 for media playback. This implementation follows HLS standard in which we utilizes an open source library named FFMPEG. A conclusion of this paper is that live steaming on mobile devices over HTTP is achieved without any plug-ins.

Keywords: *HTML5, HLS, IP Camera, FFmpeg*

1. Introduction

Analog Closed Circuit Television (CCTV) cameras have been widely used for applications of security surveillance. In recent years, the development of internet communication and rapid changes in technology have resulted in the advancement of a new surveillance system, which employs a digital internet protocol (IP) camera for real-time monitoring and surveillance at any time and from any place. In other words, information transmission technologies have made a paradigmatic shift from a traditional text-centric approach to a multimedia based approach. An IP camera decodes and compresses an input image and transmits it to a user through a wired or wireless network. Studies have been conducted on multimedia transmission methods that employ the live streaming protocol for providing information to users in real-time [1-11].

There are three main methods to deliver multimedia: traditional streaming, progressive download, and adaptive streaming. Traditional streaming requires a stateful protocol which establishes a session between the service provider and client. In this technique, media is sent as a continuous stream of packets over UDP or TCP transport. The Real-Time Transport Protocol (RTP) together with the Real-Time Streaming Protocol (RTSP) are frequently used to implement such service [9, 10]. In contrast, HTTP is stateless. Progressive download is a technique to transfer data between server and client from standard HTTP Web servers. It has become very popular and it is widely used on the Internet. Users request multimedia content which is downloaded progressively into a local buffer. As soon as there is sufficient data the media starts to play. If the playback rate exceeds the download rate, then playback is delayed

* Corresponding Author

until more data is downloaded. Disadvantages of progressive download are mostly that bandwidth may be wasted if the user decides to stop watching the content after progressive download has started, it is not really bitrate adaptive and it does not support live media services. Adaptive streaming is a technique which detects the user's available bandwidth and CPU capacity in order to adjust the quality of the video that is provided to the user, so as to offer the best quality that can be given to this user in their current circumstance. It requires an encoder to provide video at multiple bit rates (or that multiple encoders be used) and can be deployed within a Content Delivery Network (CDN) to provide improved scalability. As a result, users experience streaming media delivery with the highest possible quality [7, 8].

Recently a new solution for adaptive streaming has been designed, based on the stream switching technique. It is a hybrid method which uses HTTP as a delivery protocol instead of defining a new protocol. Video and audio sources are cut into short segments of the same length. All segments are encoded in the desired format and hosted on a HTTP server. Clients request segments sequentially and download them using HTTP progressive download. Segments are played in order and since they are contiguous, the resulting overall playback is smooth.

Apple released a HTTP-based streaming media communication protocol called HLS [1, 2] to transmit bounded and unbounded streams of multimedia data. According to this specification, an overall stream broken into a sequence of small HTTP-based file downloads, where users can select alternate streams encoded at different data rates. Initially, users download an extended M3U playlist which contains several Uniform Resource Identifiers (URIs) [14] corresponding to media files, where each file must be a continuation of the encoded stream. Each individual media file must be formatted as an MPEG-2 transport stream (TS) or a MPEG-2 audio elementary stream. Currently, iOS/Safari is the only platform with built-in adaptive streaming, supporting Apple's own HLS protocol based on HTML5. Android introduced HLS support since the latest version 4.0 [12, 15]. Dynamic Adaptive Streaming over HTTP (DASH) also addresses the weaknesses of RTP/RTSP-based streaming and progressive download [3, 4].

IP camera live streaming is made possible by a client-server structure. Almost all commercial IP cameras employ the RTP/RTSP protocol in order to support live streaming [9, 10 and 18]. In that case, a user is required to install a separate application program or download plug-ins supporting RTP/RTSP from the server to transmit multimedia data.

In recent years, innovations in technology have enabled the reproduction of multimedia in a browser using standardized HTML without using a separate plug-in or a dedicated application program. The latest version of HTML, HTML5, is currently under developing standard; specifically, web pages that are based on HTML5 are able to provide graphics and multimedia to users without employing a separate plug-in. Furthermore, to provide consistent content and service to users, HTML5 developers have taken various developments and utilization environments of users into account and continue to successively upgrade its features [12].

One major problem encountered in the support of HLS is achieving IP camera live streaming in an Apple device without employing a separate plug-in. Therefore, the goal of this study is to realize live streaming of HLS through an open source library in an embedded system programming environment for a commercial IP camera. HTML5 first describes new tags for multimedia, which are the new audio and video tags. We produced an implementation of HLS for the IP camera by using elements of HTML5 for media playback.

Generally, realization of live streaming requires the IP camera to export images in the form of TS files, which are segmented in real-time. Almost all commercial IP cameras produce image data in Joint Photographic Expert Group (JPEG) format or another similar image format. Image format is defined according to company preferences. The generated raw image data is converted using standard image compression methods and is transmitted to users through the client-server architecture. The IP camera images are generally displayed through management software in the security surveillance, which employs multiple IP cameras. This study aims to realize live streaming in a standard browser by supporting the HLS protocol. Compatibility for supporting the existing method is maintained without employing separate plug-ins or management software.

In this paper, a commercial IP camera, model FW1173-DS, is used [18]. For supporting the HLS protocol, an open source FFMPEG library is used as a media converter [17]. FW1173-DS generates image data in the format of JPEG Elementary Stream (JES). JES image data is encoded by H.264 and is adapted in security surveillance systems in management software through a transmission protocol. In this paper, we generates an MPEG-2 TS file which is the standard for the HLS protocol using JES image data while supporting compatibility to the existing surveillance. First, JES image data is converted to a raw H.264 stream and the raw H.264 stream is then converted to segmented MPEG-2 TS files by a media converter of FFMPEG library. And an M3U playlist file is produced and then live streaming is achieved to the client through a web server.

In Section II, we propose an HLS Architecture for supporting compatibility with commercial IP cameras which are used in the security surveillance systems. Section III explains process of implementing the HLS Protocol and building a stream segmenter, media converter, playlist generation, and web server. Sections IV and V describe the experimental results and conclusions, respectively.

2. HLS Architecture for a Commercial IP Camera

Live streaming for the IP camera generally requires a plug-in or a live streaming protocol supported in a web browser. Table 1 shows a comparison of streaming services currently being used. Currently, live streaming protocols are realized by using Adobe Flash Player or Microsoft Internet Information Services (IIS) plug-ins. In order to avoid using a plug-in, an IP camera should be implemented with the HLS or DASH protocols. Of these two options, we chose to implement HLS.

This paper aims to achieve live streaming in mobile devices for an IP camera, generally used in security surveillance, without installing a plug-in or a separate application program. HLS is a protocol that supports HTML5 standards and has a feature of no client dependency. Furthermore, the system architecture is implemented to support the HLS protocol while maintaining compatibility with existing IP cameras that are used in dedicated management software.

Table 1. Comparison of Streaming Services [3-8]

Feature	Apple	Adobe	MS IIS	MPEG-DASH
On-Demand & Live	Yes	Yes	Yes	Yes
Adaptive bitrates	Yes	Yes	Yes	Yes
Delivery Protocol	HTTP	HTTP	HTTP	HTTP
Origin Server	Web Server	Adobe Media Server	MS IIS	Web Server
Media Container	MP2 TS	MP4-part 14, FLV	MP4-part 14	MP2 TS, Fragmented MP4
Video Codecs	H.264 Baseline Level	H.264	Agnostic	H.264, SVC, Multiview Coding,

				MPEG 4 AAC
Default Segment Duration	10 seconds	2 seconds	4 seconds	Flexible
End-to-End Latency	30 seconds	6 seconds	>1.5 seconds	Flexible
File Type on Server	Fragmented	Contiguous	Contiguous	Fragmented
Client Dependence	No	Flash Player	Silverlight	No

As shown in Table 1, HLS is a protocol implemented for Apple's iOS/Safari, QuickTime Player, and Apple TV to provide live streaming and video on demand (VOD) streaming [1, 2]. The ability of HLS supporting Android devices in addition to Apple devices is suitable as a live streaming protocol for mobile devices

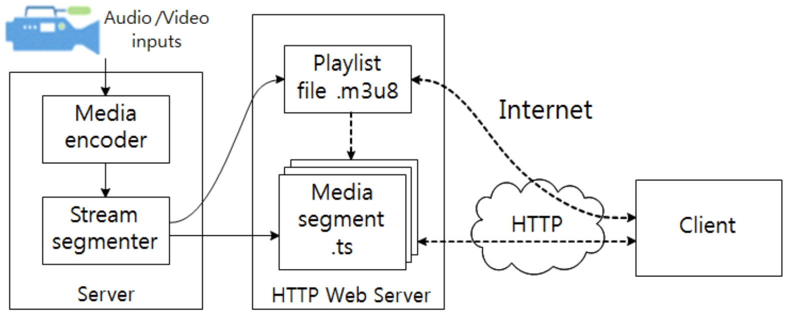


Figure 1. System Architecture of http Live Streaming Protocol

Figure 1 shows the general structure of HLS Protocol. After the media is encoded, it is segmented through a stream segmenter. The stream segmenter segments the media data, which is received with respect to a fixed time interval, generates a segmented file, and then generates a playlist file of meta data (.m3u8) through which the segmented file can be accessed from client devices. Live streaming requires storing data in real-time; thus, segmenting media data through a segmenter is not required. Instead, segmented streaming data can be produced based on a predetermined duration of a media segment which is then stored. The playlist file, which has an m3u8 file extension, has at least three types of TS files in MPEG-2 media format. Users download an extended M3U playlist file which contains several URIs corresponding to media files. Each individual media file must be formatted as an MPEG-2 TS. For continuous live streaming, the playlist must be updated with the production of the MPEG-2 TS file together with the produced URIs.

In this paper, to maintain compatibility with the management software of the existing security surveillance system, the sequence of the media encoder and stream segmenter is inverted as shown in Figure 2. Moreover, the structure is configured to support the HLS protocol with JES image data encoded by an IP camera. Thus, the structure is designed to maintain compatibility with the integrated management software of existing security surveillance IP cameras that use JES image data.

Realization of IP camera live streaming based on the HLS protocol requires a server to generate media files in real-time. This is followed by real-time update of the playlist file. The stream generator of the HLS Protocol generates raw H.264 stream data with a predetermined duration of a media segment using JES image data of Audio/Video as input to the camera. The media encoder generates a segmented media file for the HLS protocol which is converting raw H.264 stream data to MPEG-2 TS format using the FFMPEG library. The generation of the M3U playlist file of the media segments makes linking to these segments possible when the play list is requested by a client.

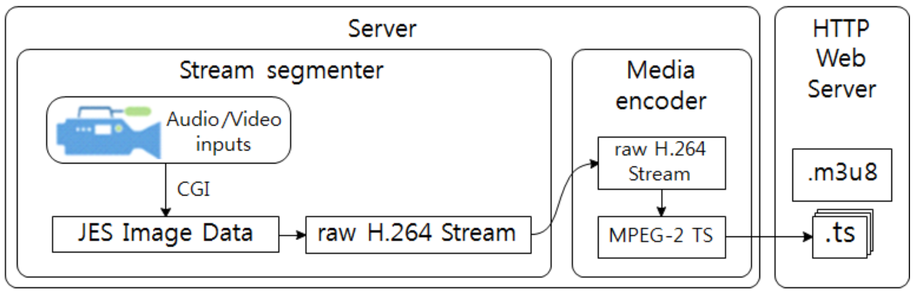
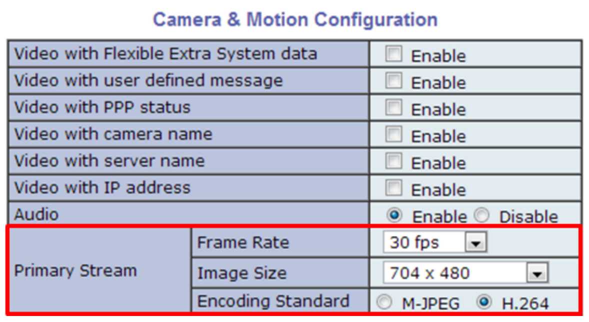


Figure 2. System Architecture of HLS for an IP Camera

3. Implementation of HLS Protocol for the IP Camera

3.1. Stream Segmenter

Figure 3 shows the configuration menu for the JES Image Data in the camera through the HTTP Web Server. Generally, image data of the IP camera is transmitted to a server through a Common Gateway Interface (CGI). In order to support the HLS protocol using the received image data in the server, it is required to do continuous generation of an MPEG-2 TS file format that is encoded using the H.264 codec. Therefore, the output stream format of FW1173-DS is chosen as the H.264 encoding format as shown in Figure 3.

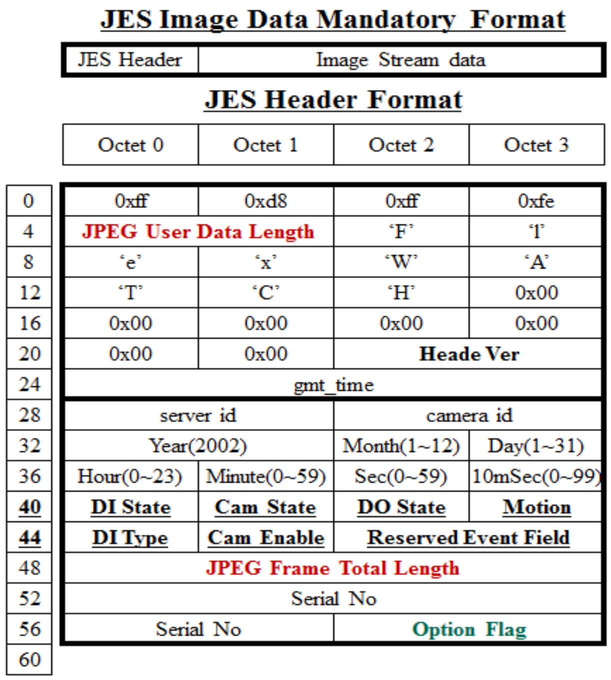


Camera & Motion Configuration

Video with Flexible Extra System data		☐ Enable
Video with user defined message		☐ Enable
Video with PPP status		☐ Enable
Video with camera name		☐ Enable
Video with server name		☐ Enable
Video with IP address		☐ Enable
Audio		[✓ Enable] Disable
Primary Stream	Frame Rate	30 fps
	Image Size	704 x 480
	Encoding Standard	M-JPEG [✓ H.264]

Figure 3. Configuration of the IP Camera of FW1173-DS

Even though the encoding standard of the output stream from the IP camera is H.264, the actual data format of output image stream is JES which is utilized in the management software for security systems. The JES image data is composed of a JES header and the image stream data. The JES Image Data Mandatory format and JES Header Format is shown in Figure 4. The JES Header consists of the frame in-formation of the image data and status information of the IP camera including vendor information. In this paper, we requires a raw H.264 stream so a raw H.264 stream file from JES image data is made by extracting only the image data stream encoded with H.264, while excluding JES header. Finally, a TS file for supporting the HLS protocol and a playlist file for playing the TS file can be obtained from the raw H.264 stream file by using an open source FFMPEG library.



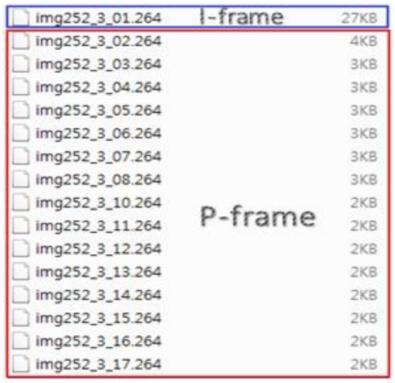
JES Image Data Mandatory Format

JES Header	Image Stream data

JES Header Format

	Octet 0	Octet 1	Octet 2	Octet 3
0	0xff	0xd8	0xff	0xfe
4	JPEG User Data Length		'F'	'I'
8	'e'	'x'	'W'	'A'
12	'T'	'C'	'H'	0x00
16	0x00	0x00	0x00	0x00
20	0x00	0x00	Heade Ver	
24	gmt_time			
28	server id		camera id	
32	Year(2002)		Month(1~12)	Day(1~31)
36	Hour(0~23)	Minute(0~59)	Sec(0~59)	10mSec(0~99)
40	DI State	Cam State	DO State	Motion
44	DI Type	Cam Enable	Reserved Event Field	
48	JPEG Frame Total Length			
52	Serial No			
56	Serial No		Option Flag	
60				

Figure 4. JES Image Data Mandatory Format and JES Header Format



img252_3_01.264	27KB
img252_3_02.264	4KB
img252_3_03.264	3KB
img252_3_04.264	3KB
img252_3_05.264	3KB
img252_3_06.264	3KB
img252_3_07.264	3KB
img252_3_08.264	3KB
img252_3_10.264	2KB
img252_3_11.264	2KB
img252_3_12.264	2KB
img252_3_13.264	2KB
img252_3_14.264	2KB
img252_3_15.264	2KB
img252_3_16.264	2KB
img252_3_17.264	2KB

Figure 5. Elementary Streams of FW1173-DS

The raw H.264 streams named Elementary Streams extracted from the JES Image data are shown in Figure 5. The generated image streams are I-frame or P-frame according to the JES Image data header information. In order to support the HLS protocol, we generates a single raw H.264 stream file by accumulating the generated raw H.264 streams as shown in Figure 5 for 10 seconds which is typical duration for TS of HLS.

3.2. Media Encoder

The media encoder converts the generated raw H.264 stream file to a segmented MPEG-2 TS file of having10 seconds stream. The open source multimedia converter, FFMPEG, is utilized to build MPEG-2 media. The design goal of media encoder is to minimize the memory size of an embedded LINUX based IP camera. To cope with this objective, we optimized FFMPEG library with using necessary functions.

The employed libraries from FFMPEG library to implement the media encoder when converting raw H.264 stream data to MPEG-2 TS format include libavformat, libavcodec and libavutil. The library libavutil supports functions to enable processes such as string processing, random number generation, special mathematical functions, and multi-media encryption programming. The library libavcodec employs an encode/decode framework to enable encoder/decoder, subtitle streaming, and a bit-stream filter operation. The library libavformat consists of a muxer/demuxer function for a multi-media container format.

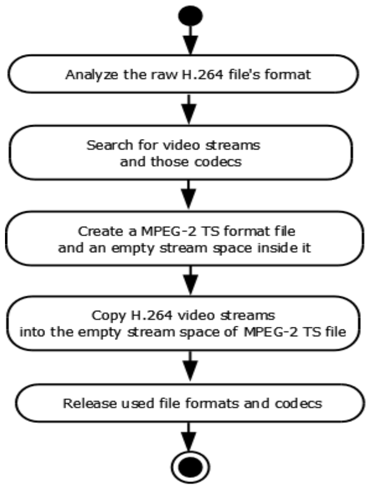

Analyze the raw H.264 file's format
Search for video streams and those codecs
Create a MPEG-2 TS format file and an empty stream space inside it
Copy H.264 video streams into the empty stream space of MPEG-2 TS file
Release used file formats and codecs

Figure 6. Procedures for Converting H.264 steam to TS File

The procedures conducted in converting the raw H.264 stream file to MPEG-2 TS file of having 10 seconds are shown in Figure 6. First, the generated raw H.264 stream file is examined to find out its file format. The analyzed information is stored at data structure located in the library of libavformat. Next, the image and codec information in the raw H.264 stream file is searched. Once a video stream is searched, relevant image codec information is examined. The codec information of the relevant image is stored at data structure using codec information of libavcodec.

The output file is subsequently generated in MPEG-2 TS format using the file creation function of libavutil. In this generated output file, an empty stream space is prepared. Codec information of the raw H.264 stream data and stream data extracted according to Section 3.1, are copied to the empty stream space; the stream data is copied by a single frame unit of H.264 format. Through a repeated process, the output file attains the MPEG-2 TS file format. However, when this process is repeated in a single processor, the spaces dynamically allocated in the heap area of memory are not returned. In this case, the operating system forces the process causing the overflow of unreturned memory to terminate. The final step shown in Figure 6 is a process to prevent memory overflow. File format information and codec information used in the file creation process are directly released through the free and close functions of libavutil.

3.3. Playlist Generation

The procedure for generating and extracting a raw H.264 stream from an IP camera in MPEG-2 TS file format is initiated at the same time as when the raw H.264 stream of having 10 sec duration of a media segment is obtained. The generated the raw H.264 stream is converted to MPEG-2 TS file format through *FFmepeg_thread*. Once conversion is complete, *FFmepeg_thread* returns the resource and modifies the playlist of the MPEG-2 TS file. Thus, by repeating the procedure, MPEG-2 TS files and playlist required for live streaming of the HLS protocol, are continuously generated.

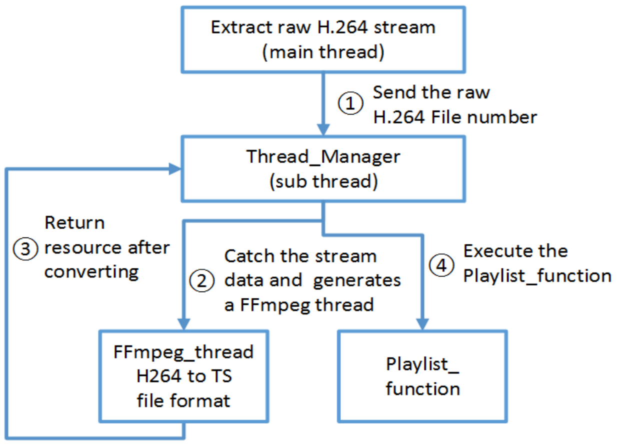

Extract raw H.264 stream (main thread)
① Send the raw H.264 File number
Thread_Manager (sub thread)
③ Return resource after converting
② Catch the stream data and generates a FFmpeg thread
④ Execute the Playlist_function
FFmpeg_thread H264 to TS file format
Playlist_ function

Figure 7. Playlist Generation Considering Real-Time Update for Live Streaming

Pseudo-code for *Thread_Manager* of Figure 7 is shown in Figure 8. The extracted raw H.264 streams are converted to TS files through employing libraries during a pre-set duration of a media segment. A segmented TS file is then generated while a playlist is simultaneously generated or modified using the Playlist function ().

```
Procedure Thread_Manager
  If Queue is not empty then
    dequeue get filenumber;
    create FFmpeg_Thread;
    execute FFmpeg_Thread(filenumber);
  If filenumber >= MAX_FILE_NUMBER then
    open playlist file
    remove Headmost Media in the playlist file;
    increase media sequence++;
    close playlist file
end
```

Figure 8. Pseudo Code of the *Thread_Manager*

The fucntion of *FFmpeg_Thread* is described in Figure 9; the raw H.264 stream is converted to the TS file format based on the duration of a media segment. And in this thread, updating playlist by adding the generated MPEG-2 TS files is achieved.

```
Procedure FFmpeg_Thread(void * arg)
  set filenumber to *((int *)arg);
  set currentfile to "video%d.ts", filenumber;
  If currentfile exist then
    remove the currentfile;
  convert file format to MPEG-2 TS from raw H264;
  add URL of converted video%d.ts into playlist file;
end
```

Figure 9. Pseudo Code of the *FFMPEG_Thread*

3.4. Web Server

A client must request the HLS server for HTTP to receive a response. In other words, the segmented media and the information of media to be played are transmitted to the client; when media playback is complete, the play list is reloaded, and the generated media is played continuously upon receiving a request.

The M3U file format uses the UTF-8 character set by conforming to the extension standards of the m3u file format. The M3U file is the playlist file containing a list of TS files, which are played continuously. Table 2 shows the basic tags that are employed.

Table 2. Important Tags of the M3U File Format (In reference [2], Chapter 3.3-3.4 Tags)

Format	Descriptions
#EXTM3U	An Extended M3U file is distinguished from a basic M3U file by its first line, which MUST be the tag #EXTM3U. This tag MUST be included in both Media Playlists and Master Playlists.
#EXTINF:<duration>,<title>	The EXTINF tag specifies the duration of a media segment.
#EXT-X-TARGETDURATION:<s>	It applies only to the media segment that follows it, and MUST be followed by a media segment URI. Each media segment MUST be preceded by an EXTINF tag.
#EXT-X-MEDIA-SEQUENCE	Each media segment in a Playlist has a unique integer sequence number. The sequence number of a segment is equal to the sequence number of the segment that preceded it plus one. This tag indicates the sequence number of the first segment that appears in a Playlist file.
#EXT-X-ENDLIST	The EXT-X-ENDLIST tag indicates that no more media segments will be added to the Media Playlist file. It MAY occur anywhere in the Playlist file; it MUST NOT occur more than once.

As shown in Table 1, MPEG-2 TS is used as the format for media files used in the HLS protocol. In this study, a file is generated to support a standardized HLS protocol by using stream data that is input from an IP camera. A video file in MPEG-2 TS File format is generated based on a pre-determined duration of a media segment. A tag, as shown in Table 2, is used in the M3U playlist file; an example of an M3U playlist file is shown in Figure 11. VOD streaming and live streaming in the play list structure can be distinguished by the presence or absence of the "#EXT-X-ENDLIST" Tag.

```
#EXTM3U
#EXT-X-TARGETDIRATION:10
#EXT-X-MEDIA-SEQUENCE: 0
#EXTINF:10.00,
http://www.example.abc/hls_test/segment0.ts
#EXTINF:10.00,
http://www.example.abc/hls_test/segment1.ts
#EXTINF:10.00,
http://www.example.abc/hls_test/segment2.ts
```

Figure 10. Example of an M3U Playlist File

The HLS server receives a file transmission request from a client through a HTTP and provides a response to the player. The requested media file is transmitted by responding to the request according to the HTTP data transfer protocol. Therefore, to implement the HLS server, a web server should be used, which can read the stored data and transmit data according to HTTP responses [1, 16].

In this study, Apache2 is used as a Web server for the HLS Server; no special configuration is necessary, apart from associating the Multipurpose Internet Mail Extension (MIME) types of the files being served with their file extensions [1]. The MIME type of the Apache2 Web server is presented in Table 3.

Table 3. MIME Types for HTTP Live Streaming [1]

File Extension	MIME Type
.m3u8	application/x-mpegURL or vnd.apple.mpegURL
.ts	video/MP2T

```
<html>
   <head>
      <title>HTTP Live Streaming Example</title>
   </head>
   <body>
      <video src="http://www.example.com/hls_test/playlist.m3u8"
                    height="300" width="400" controls></video>
   </body>
</html>
```

Figure 11. Serving HLS Protocol in a Webpage

The HTML of a web page, used for serving the TS File to the client by using Apache2 web server, is represented in Figure 11. HLS is supported by the HTML5; thus, when linking playlist.m3u8 using the <video> tag, the client can play media in a standard web page through the URL written in the playlist using the HLS protocol.

4. Result of the HTTP Live Streaming Play Test

Experiments for HLS playback were performed on iOS 7.0.4 (iPhone 4S), Android 4.4 (Nexus7) and Android 4.3 (Galaxy S4). As shown in Figure13, the streaming data for the IP camera is played in real-time on an iPhone without any plug-ins. The real streaming data has delay of 30 seconds because the real-time streaming can be started after the playlist have 3 MPEG-2 TS files which have 10 seconds duration each. This feature is summarized in Table I as End-to-End Latency of 30 seconds for HLS protocol. HLS protocol is also supported in an Android device which has Galaxy S4 browser; however, a simple problem is encountered in the Android devices [12, 15]. Playback is possible in Android devices when it is done through additional media playback application programs such MXplayer or DicePlayer. MXPlayer is installed in Android machine as a default media player. Figure 14 shows live streaming demonstration in Android 4.3 with MXPlayer. We expect that Android will support HLS protocol [15] then live streaming is achieved on Android machines without playing any multimedia players.

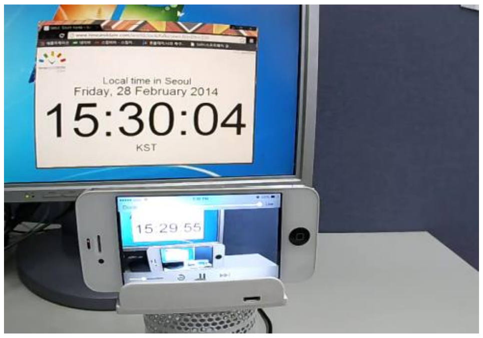


Figure 12. Live Streaming of HLS on iOS7 without any Plug-ins

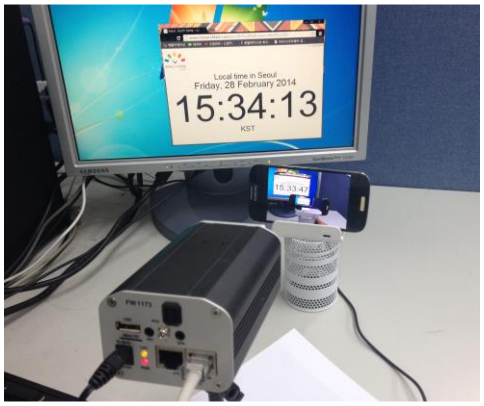


Figure 13. Live Streaming of HLS on Android 4.3 with MXplayer

5. Conclusions

HTTP based live streaming has the better traversal of NAT and firewalls, ease of deployment and built-in friendly bandwidth sharing. One of adaptive streaming approaches over HTTP is Apple's HLS. This paper details a method for realizing HLS for a commercial IP camera since IP cameras are major devices for security surveillance system.

This implementation follows the MPEG standard and HLS protocols so client device is able to play live stream data on Apple devices without any plug-ins. This paper utilizes FFMPEG library to use the transcoding capabilities when making MPEG-2 TS files. FFMPEG library supports many container formats include MPEG-2 TS and raw H.264. Generally, the commercial IP cameras produce raw image data encoded with H.264 format and HLS is based on MPEG-2 TS files. Therefore, FFMPEG library is useful to implement HLS protocol and the function of FFMPEG is ported into the IP camera as an embedded system program.

In order to generate live streaming continuously, we developed a thread program to update M3U playlist files which contains three TS files of 10 seconds duration. Based on HTML5 in the Web server, live media was played on the client by using an Apple Device without any kinds of plug-ins. For Android devices, we have to use a default media player. Results showed that 30 seconds latency due to the standard of HLS which have three TS files prior to send media data to the client.

Acknowledgements

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

References

[1] Apple Inc., "HTTP Live Streaming Overview", iOS Reference Library, **(2014)**.
[2] R. Pantos, W. May and Apple Inc., "HTTP Live Streaming: draft-pantos-http-live-streaming-12", IETF draft, **(2013)** October.
[3] L. R. Romero, "A Dynamic Adaptive HTTP Steaming Video Service for Google Android", MS Thesis, KTH Information and Communication Technology, **(2011)**.
[4] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP - Standards and Design Principles", ACM Multimedia Systems Conference (MMSys), San Jose, California, USA, **(2011)** February 23-25.
[5] G. J. Yang, B. W. Choi and J. H. Kim, "Implementation of HLS Protocol for an IP Camera", AST Letters. Networking and Communication, Jeju Island, Korea, **(2014)** April 15-18.
[6] A. Biernacki and K. Tutschku, "Performance of HTTP video streaming under different network conditions", Multimedia Tools and Applications, **(2013)**.
[7] C. Knowlton, "Adaptive Streaming Comparison: The Official Microsoft IIS", http://www.iis.net/learn/media/smooth-streaming/adaptive-streaming-comparison, **(2010)**.
[8] Microsoft Corporation, "ISS Smooth Streaming Transport Protocol", **(2009)**.
[9] H. Schulzrinne, U. Columbia, A. Rao, R. Lanphier Netscape and RealNetworks, "Real Time Streaming Protocol (RTSP)", RFC 2326, IETF draft, **(1998)**.
[10] AVT Working Group, H. Schulzrinne, GMD Fokus, S. Casner, Precept Software, Inc., R. Frederick, Xerox PARC, V. Jacobson and LBN Lab, "RTP: A transport protocol for real-time application", RFC 1889. IETF draft, **(1996)**.
[11] Adobe Systems Inc., "Real-Time Messaging Protocol (RTMP) specification", **(2009)**.
[12] J. Wijering, "The State of HTML5 Video", http://www.jwplayer.com/html5/, **(2013)**.
[13] X. Yan, L. Yang, S. Lan and X. Tong, "Application of HTML5 Multimedia", International Conference on Computer Science and Information Processing, Xi'an, Shaanxi, China, **(2012)** August 24-26.
[14] T. Berners-Lee, MIT/LCS, R. Fielding, U. C. Irvine, L. Masinter and Xerox Corporation, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, IETF draft, **(1998)**.
[15] Android Devices Supported Media Formats, "http://developer.android.com/guide/appendix/media-formats.html".
[16] Apache Software Foundation, "http://www.apache.org/".
[17] FFMPEG, "http://FFMPEG.org/".
[18] Seyeon Tech., "http://www.flexwatch.co.kr".