Adaptive Transcoding HTTP Live Streaming System

Lain-Jinn Hwang and Chien-Yi Lu
Department of Computer Science and Information Engineering, Tamkang University
Email: 700410193@s00.tku.edu.tw

***Abstract*—The most popular Internet live streaming technique, Apple HTTP Live Stream, which causes that users suffer from the consistency in bitrate when the bandwidth between the user and the server varies. Thus, this article solves the issue mentioned previously. The method we conduct makes use of adaptive transcoding technique and bandwidth management approach in order to achieve the goad that the video stream quality varies in according to the bandwidth change. We have the stream fluency the major consideration while the maximum bandwidth utilization the minor to assure the highest video quality.**

***Keywords*—HTTP Live Streaming, Mobile Streaming, Bandwidth Detection, Adaptive HTTP Streaming, WebVTT.**

I. Introduction

WITH the fast spreading of Internet connections and increasing landlines and mobile bandwidth for data upload and download, network based multimedia services are getting diversified every day. Take network streaming for example. From the network broadcasting in the early days to the HD video streaming now, users are not using devices for page browsing only. A recent research report suggests that more than half of bursting global traffic can be attributed to network streaming [1][2]. Thanks to the popularization of smartphone and tablet computer, a full range of cloud services is now true. This adds more diversified network streaming platforms. Cloud services have shifted from limited indoor locations on landline networks in the early days to browsing anywhere you go, either indoor or outdoor [3]. Ever the larger screen size and higher resolution of smartphone and tablet computer enables viewers to enjoy better quality program [3]. The Internet connection traffic by smartphone in Taiwan is soaring. Latest mobile network bandwidth tests indicate that network connection in Taiwan is relatively unstable with speeds varying at different locations. The worst place for networking is on a moving MRT. Time is another factor in determining communication speed with smartphone. Average phone connection speed is relatively better from night to before working hours and worse from working hours to one o'clock in the morning. Despite the adoption of LTE high speed network, the installation is still in process and people in Taiwan may still suffer limited bandwidth. Before the mobile network system is well set up in Taiwan, browsing the Internet or watching contents streamed by it may leave you poor experiences. Outdoor networking bandwidth varies more significantly than in an indoor environment. The strength of local signal, number of users connected to the base station, local bandwidth and changes in bandwidth caused by movement all may impact online streaming. Any environment with unstable bandwidth is worth studying, ensuring users' optimum viewing quality by balancing screen definition and fluency [4][5].

Conventional network streaming mandates the installation of the same program in both server and client end computers. The server end compresses contents with fixed bit rate (image quality) and the same encoding format and sends both video and audio contents to the client end through TCP, UDP or RTP/RTCP protocols. The client end computer then decodes and plays back received streaming contents. With fixed bit rate compression and pre-encoding, it can easily provide high definition video in case of sufficient bandwidth. However, when playing this kind of video in a network with limited bandwidth, viewers suffer from continuous buffering status. As more people are connected through mobile network and live in a fast moving mode, they require fast information retrieval and instant services rather than waiting for the download of films. This study tries to solve this problem by providing users with instant viewable HD streams without encoding videos in a supported format in advance.

A lot of network platforms require online stream viewers to install player software that supports the desired streaming services before viewing contents offered by them. Users are required to select a player program but not all of them have this capability. A better way of selecting the proper program to view streams should be provided instead of asking users for player selection on the basis of format of contents to be played. User experiences can be improved a lot if their operation can be simplified if required services can be accessed by opening their browsers.

II. Background

The HTTP Adaptive Streaming (HAS) technology combines the features of conventional streaming technology and HTTP's progressive download and playback to deliver media contents in HTTP. The HAS improves users' media playback experiences while reducing technology complexity at the server end. This makes it adopted as the trend in video streaming development.

HAS technology integrates features of conventional RTSP/RTP streaming media technology and HTTP's progressive download to be of high efficiency, expansion, and compatibility.

The HAS technology is a mixed media transmission. The contents may look like being streamed to viewers. In reality, it delivers contents by employing HTTP protocol's progressive downloading. Media contents are divided into a series of media blocks for pull-based transmission. That is, each media block is transmitted only at the request by the client end computer. Its core technology lies in slicing videos into fixed play time

block, usually 2-10 seconds. At the video encoding layer, this implies that every slice is composed of certain number of complete video GOPs, i.e., every slice contains one key frame, to ensure independence of every slice from its last and next counterpart.

Media slices are saved in HTTP Web server. The client requests slices from the server in linear manner and download them in HTTP. Once the media slices arrived at the client they are played back in sequence. As streamed slices are encoded in given conventions there is no lost or duplicated content. This ensures viewers seamless and smooth playback of streaming contents.

In case a copy of the contents are encoded with multiple bit rates, then the contents slicing module can cut it into slices of individual bit rates. As the Web server streams by making the most of available bandwidth without traffic control mechanism, the client then can choose to download larger or smaller media slices for bit rate self adaptiveness by detecting valid bandwidth from the source Web server.

The core technology of HAS is composed of two parts. One is content preparation including the transcoding platform that supports multiple terminal devices and the media slicing module. The other is the streaming for content delivery on the basis of HTTP based media source server and the terminal equipments oriented contents delivery network.

A. Apple HTTP Live Streaming

Conventional network streaming mandates the installation of the same program in both server and client end computers. The server end compresses contents with a fixed bit rate (image quality) and the same encoding format and sends both video and audio contents to the client end through TCP, UDP or RTP/RTCP protocols. The client end computer then decodes and plays back received streaming contents. A better way of selecting the proper program to view streams should be provided in an era of cloud instead of asking users for player selection on the basis of the format of contents to be played. User experience can be improved a lot if their operation can be simplified and if required services can be accessed by opening their browsers.

Conventional RTP protocol based online streaming technology employs a special protocol and communication port and makes it easy to be blocked by firewall unless the port is exclusively opened for it. The design requires it to maintain being connected throughout the entire playback session. The server end is then required to manage every connected client with a single and exclusive streaming session. This consumes a lot of resources at the server end. On the contrary, the HAS technology requires the server end to process packets in sequence by employing HTTP protocol to deliver them. This not only reduces server end workloads but also avoids being blocked by a firewall as the communication port required by HTTP protocol is the one that must be opened for web server.

Despite enabling users to view video streaming through a browser, Adobe Flash Dynamic Streaming and Microsoft Smooth Streaming suffers from the inconvenience of plug-in installation, as neither are supported by the browser itself.

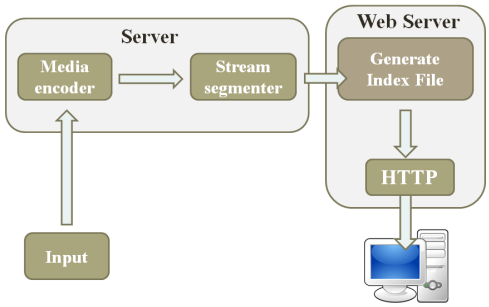

Fig. 1: HLS architecture

This is not user friendly. As mobile devices, smartphones and tablet computers are getting popular everyday and most do not support plug-in added in their browsers, most network streaming is viewed in a conventional manner where a special application is installed to keep the connection with the server for video transmission. Apple HTTP Live Streaming is few of the systems that support network stream viewing in a browser.

Differing from other RTP/RTSP online streaming technology, the HLS employs HTTP protocol rather than a customized one. This helps in avoiding to be blocked by firewall or proxy server. The HTTP protocol's TCP data transmission mechanism also ensures A/V quality and correctness. The HLS system also has merits in that it enables you to set up a VOD online streaming system with any popular web server in a fast and easy way. That is why this thesis selects HLS as the technology to implement the theory.

HTTP Live Streaming employs HTTP protocols. This comes with two benefits. The first is that video data can be delivered by a browser such that service pages can be revised without any need of software installation at the client end. The second is that it is less likely to be blocked by a router firewall. Viewers can access network streaming with browser behind routers without changing router's firewall rules.

See Figure 1 for the structure of HTTP Live Streaming. Once a video clip arrives, it is compressed and encoded into a MPEG2-TS before cutting into short and small Transport Streams of 10 seconds length, recommended by Apple Inc. A playlist of M3U8 format is then generated that contains URL of every Transport Stream. Viewers then can play back the video by opening the M3U8 playlist in their browser. Embedding the M3U8 playlist in an HTML5 page may ease the playback even further. The HTTP Live Streaming offers multiple definition formats with the M3U8 playlist. Viewers may select proper playback definition based on available bandwidth and image quality data stored in the M3U8 index file.

The basic theory is to slice one Transport Stream (TS) into many pieces of short time span streaming files. After one streaming file is downloaded, the streams starts to provide to the user. As each TS is short and small, users can start their viewing quickly with reduced streaming delay. In addition, when playing online streaming in browser, the server end delivers multiple TSs with different definitions and bit rate for the browser. The latter then selects proper definition based on

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:10,
http://url/segment0.ts
#EXTINF:10,
http://url/segment1.ts
#EXTINF:10,
http://url/segment2.ts
#EXT-X-ENDLIST
```

Fig. 2: M3U8 Format

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000
http://ALPHA.mycompany.com/lo/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000
http://BETA.mycompany.com/lo/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
http://ALPHA.mycompany.com/md/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000
http://BETA.mycompany.com/md/prog_index.m3u8
```

Fig. 3: M3U8 index file

```
WEBVTT

00:11.000 --> 00:13.000
<v Roger Bingham>We are in New York City

00:13.000 --> 00:16.000
<v Roger Bingham>We're actually at the Lucern Hotel, just
      down the street

00:16.000 --> 00:18.000
<v Roger Bingham>from the American Museum of Natural
     History
```

Fig. 4: WebVTT subtitles

```
    <video width="640" height="480" controls>
    <source src="video.mp4" type="video/mp4" />
    <source src="vidoem.webm" type="vidoe/webm" />
    <track src="subtitles.vtt" srclang="zh" label="
     Chinese" />
    </video>
```

Fig. 5: The formats of WebVTT

available bandwidth. This helps users in maintaining playback fluency in spite of changing bandwidth. To play video or audio files from browser through HLS, an M3U8 file must be added in HTML5. This is how an online streaming can be played with M3U8 file.

B. M3U8 Playlist

To enable a browser to play TS, with format as shown in Figure 2, in sequence, the tag EXT-X-TARGETDURATION is set up each TS's play time contained in the playlist. The tag EXTINF set up length of individual TS by connecting to its URL.

HTTP Live Streaming provides multiple playback definitions with a index file, see Figure 3, The tag EXT-X-STREAM-INF sets up relevant M3U8 playlist based on available bandwidth and enables the client end to choose videos.

HTTP Live Streaming is employed by video on demand services where the film is pre-encoded and saved in the server end. It adjusts the time line to the desired time point for playback without having to wait for reading the entire file before playing it back. This reduces viewers' waiting time and improves viewing experiences. However, if the input video is a real time image, e.g., webcam, then only one output format can be generated immediately. Here the client end may not select playback definition in accordance with available bandwidth and the M3U8 index file.

C. Subtitle file (WebVTT)

With HTML5 support, a browser is now able to play videos without installing plug-ins. However, without the help of subtitles viewers may lack the ability to understand videos in a foreign language. The HTML5 video subtitle, defined in WebSRT format, exists as part of HTML5 standards to display subtitles via general SRT document. It was renamed WebVTT later with subtitle standards originated from the HTML5 one. It is a standards system now[6][7].

With extension ".vtt" and UTF-8 code, WebVTT is a plain text file, see Figure 4, which supports multi-language subtitle display. It contains two types of information. One is the timeline and the other the subtitle texts. It begins with the WEBVTT markup trailed with a carriage return symbol. The timing reminder is in the format of HH:MM:SS.sss. The beginning and ending reminder is composed on one blank space, two hyphens, and one greater than symbol and separated with another blank space. The timing reminder starts with a new line and ended with a carriage return symbol. The accompanying subtitle text follows. The subtitle texts may take one or more lines without any blank line in between. The WebVTT file's MIME type is "text/vtt".

As shown in Figure 5, the WebVTT uses track tag to insert vtt subtitle file to combine HTML5 to output subtitle texts in video streams. The time line contained in the subtitle file works together with its counterpart in the streaming file to synchronize subtitle texts with the video contents [6]. As the HTTP Live Streaming slices videos into huge amount of short time span clips, the original WebVTT does not fully applicable with HLS. The time line in a single subtitle file cannot synchronize with all video files. Here individual WebVTT subtitle file is required for every video file to display WebVTT subtitle file in HLS streamed film [8].

III. Adaptive Transcoding HTTP Live Streaming System

Video on demand system enables viewers to watch pre-recorded and processed films. It enables users to jump to a

desired position in the film at the cost of huge storage spaces for keeping all the video files provided by the server. It also leads to paused playing and poor user experience when a lack of adequate bandwidth occurs during playback.

Real-time live streaming is aimed at providing viewers with instant input signal based webcasts. It enables viewers to watch instant events, e.g., online webcasting of basketball games. The drawback of it is paused images or heavy lag time when the game is going on. Both may dampen users' viewing experiences. The most critical challenge to instant streaming is to reduce image lag time and to allow re-play during the game. This saves storage space at the server end at the cost of server's computing power for huge amount of code conversion and fluctuating number of connections. The most difficult part is its constant image output bit rate. This may halt the entire streaming when there is a lack of bandwidth.

Both approaches lead to similar difficulties. That is, the playback fluency is vulnerable to changing bandwidth and number of active concurrent viewers. This paper aims to give a model that fits current network environment and capable of dealing with conventional issued described herein.

Standard HLS fits the role of video on demand services. To enable a browser to support a dynamic regulating video bit rate the server end must prepare multiple video files of different bit rates for the same film and set up conditions in the M3U8 file to guide the browser to select, download, and play proper video file based on available bandwidth such that optimum video quality can be reached [9]. An M3U8 file that lacks bandwidth criteria may select proper video quality on the basis of browser's bandwidth regulation algorithm. As this algorithm is a "greedy" one it tends to result in an undesired outcome. Whenever a new connection is established, the "greedy" algorithm lets each connection get the maximum definition settings. If the server end bandwidth fails to let every connection keep its current image quality, every browser will select to set its image quality to the minimum level provided by existing streaming. At the next playback interval (10 seconds), every connection upgrade its quality to the next higher level. If existing total bandwidth still fails to meet the needs, all the connections go back to the bottom level again or stay at the level and wait for the next 10 seconds to upgrade to the next higher level. This process of test and setup may go on and affect the viewing quality experienced by every connected user. [10] tries to solve this problem by adding one bandwidth regulating server in between the server and client ends. When a new connection is established, the bandwidth regulating server identifies every connection's best video quality at existing bandwidth then starts its playback service. When the newly added connection starts playing, all other connections are in their best image quality setting and have the issue solved.

However, [10] leads to some new issues as well. The first is that is applies to VOD services only as only the VOD system can provide all its videos' image quality settings for the bandwidth regulating server to derive the best video quality of each connection. The second is that the bandwidth detection method is not presented in detail. Two types of bandwidth must be taken into account. The one is the total bandwidth provided by the server and the other the maximum bandwidth of both the server and clients or the maximum bandwidth for the clients' networking. If the optimum bandwidth given by the bandwidth regulating server outruns the maximum connection bandwidth available at the client end, it may still delay streaming for the clients' playback.

This paper tries to solve the problem herein with the Adaptive Transcoding HTTP Live Streaming System (AHLS) with its theoretical foundation and practices described in the following.

A. Bandwidth detection

HTTP adaptive streaming comes with two key points: To provide optimized image quality and reduce lag time from image resources arrived at the server throughout to the client end for playback. This is an end-to-end(e2e) delay[11]. The standard HLS design requires to slice video files into fixed length streaming files before sending to the client. This differs from RTSP structure's compression-when-streaming approach and so is doomed of certain e2e delay. The key is to detect bandwidth without lengthening e2e delay. This paper tries to add one virtual middle layer (VML) between the server and client end to replace the connection between them. When a client requests to download HLS streaming files, it is the VML that connects with and transmit streaming to the client and to find out the transmission time of each streaming slice. With a given size of each streaming slice ε_s and streaming transmission time τ_0, we can get the existing bandwidth β between server and client ends with formula 1.

$$\beta = \frac{\varepsilon_s}{\tau_0}, \tag{1}$$

B. Dynamic trascoding algorithm

The standard HLS structure cuts streaming into many short video streaming files. In case of fluctuating network bandwidth, it changes image quality at the next slice to regulate the streaming bit rate. Based on method proposed in this paper, we can find out existing bandwidth through bandwidth detection to compress and slice streaming output file at the server end instantly. As every connection's current bandwidth is known, it can be used as the key parameter for streaming compression to ensure the streaming file bit rate at next time point would not exceed available bandwidth.

C. The encoding compression model

The existing bandwidth can be measured by recording streaming file's download time. When the client end starts playing back the streaming file, we can use the idle time, and the server end may use the measured bandwidth for next streaming file's compression parameter.

The transcoder has two functions. The one is to encode and the other decode. For source video not in format of HLS's genuine supported MPEG2-TS or MPEG4-TS (H.264), it must be decoded into non-compressed format before being encoded into streaming file. This chapter tries to give two transcoder prototypes. The HLS is designed to encode one TS file for

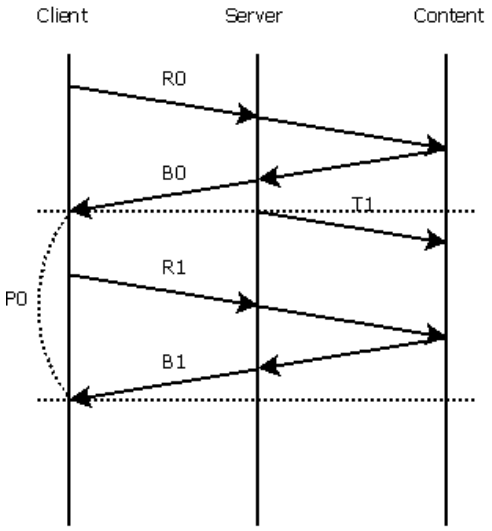

Fig. 6: Encoding model(first version)

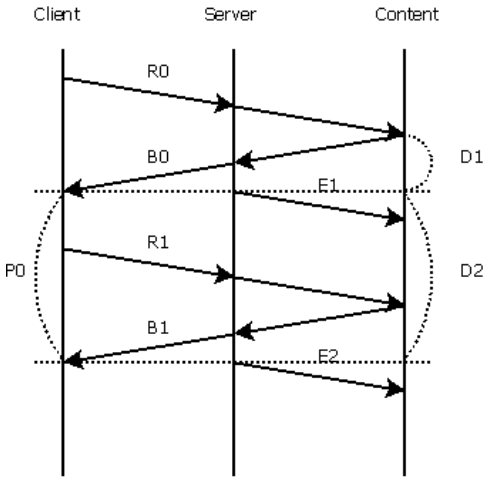

Fig. 7: Encoding model(second version)

TABLE I: The parameter of encoding model(first version)

R_0	Request first slice
B_0	Response first slice and detect bandwidth
T_1	Using bandwidth to request transcoding
P_0	Play slice
R_1	Request next slice
B_1	Response next slice and detect bandwidth

TABLE II: The parameter of encoding model(second version)

Parameter	Description
R_0	Request first slice
B_0	Response first slice and detect bandwidth
D_1	Decode first slice before
E_1	Encode first slice
P_0	Play slice
R_1	Request next slice
B_1	Response next slice and detect bandwidth
D_2	Decode second slice before
E_2	Encode second slice

every ten seconds and send the next TS file to the client when the previous one is being played back.

When sending video file to the client end, the model measures changes in bandwidth based on the download time. Figure 6 and Table I indicates that every streaming file starts to be compressed only after the previous transmission is completed. The browser requests the next streaming file when last TS file has been played for half of the time (5 seconds). The time available for encoding next streaming file is also half of the video length (5 seconds). This may lead to a problem. That is, video of higher quality may be unable to be processed here if the time required for compression is greater than 5 seconds. In case the server failed to compress next streaming file in limited amount of time, subsequent streaming download and playback may be delayed and the client end may wait for another queue time (10 seconds) before requesting next streaming file download. This may result in delayed viewing and poor viewing experience.

The second transcoder prototype tries to overcome this with faster transcoding. As shown in Figure 7 and Table II, the second prototype split the function of a transcoder into encode and decode. The server starts decoding the next streaming file before the last streaming file has been downloaded successfully. As decoding does not wait for the completion of bandwidth detection, the server end starts encoding next streaming file when the client end finished downloading the streaming file and detected existing bandwidth as well as starts playing the streaming file. This differs from the first protocol in giving ten seconds for the transcoder's necessary operation as only encoding into the streaming file is required between the bandwidth is measured and the client requests for the next streaming file.

IV. Implementation

With the popularization of mobile devices, users are getting used to browsing the Internet and watching streaming contents by connecting to mobile or wireless networks. However, mobile network's transmission speed is subject to many external factors that may lead to unstable streaming and random pauses which, in turn, dampens viewers' experience and cannot meet the fast video viewing.

To watch online streaming with mobile devices requires the installation of players to connect to the server for playback in most cases. Wouldn't it better and more convenient for viewers to play films with a browser? This helps streaming service provider to focus at web page coding and streaming service provision instead of worrying about platform compatibility and App coding.

In this section I'll outline, in directions addressed below, the software to be used, how to apply them in this paper's practice as well as potential issues, and how to solve them for

each practice presented here. I'll try to improve all the issues encountered to give designs that meet users' requirements.

A. System environment

The system environment is set up as illustrated below:

1) Install web server that supports PHP. Take this practice. It installs Apache Web Server along with PHP module.
2) Set up database system and create relevant tables. Take this practice. It installs the MySQL database in the system.
3) Copy practice program code to the default directory in web server

Once all the installations are completed, the file list database is updated and the system is ready to provide users with desired videos for viewing by saving videos to be streamed in the video folder. A cross platform online streaming player system is made by author of this paper for playback on mobile and computer devices including iPhone, iPad, any Android (4.0 or later) devices and Mac. Users may switch to full screen playback at any time, either mobile devices or computer equipment, for the best viewing experiences.

A web server is required in this paper's example. The beauty of this design is that it fits with any popular web server available now. This paper employs the most popular Apache web server available with the Linux platform along with PHP plug-in to support PHP web pages. The practice program is PHP structure based. It interfaces with the client end in PHP coded web pages. PHP is a descriptive language that can code program to run at the server end.

B. System flow

This paper coded two PHP programs that run on the server end. It brings no burden on the client end and runs only when the client end requires it. The first PHP program dynamically generates an M3U8 file when the client end requests a streaming file download. The second one sends a streaming file to the client end and derives current bandwidth based on consumed transmission time after the transmission is completed. The bandwidth data then is passed into FFmpeg to set up streaming quality for the next streaming slice.

See Figure 8 for operation flow of the program (first edition). Here, the user selects his/her desired videos in the playback interface, the client end then retrieves a new M3U8 playlist from the server end, the client end then determines the way to and starts to download streaming file based on the M3U8 file received, it then starts playing the video file after the streaming file is downloaded. The client end repeats the process to stream for the viewer.

Native HLS's M3U8 playlist is generated in advance. What the HTML5 needs to do is to read and load the M3U8 playlist as shown in Figure 9. This is ideal for VOD environment but not this paper. This paper is aimed at instant online streaming and requires a format that supports HLS online streaming. Before the download of every streaming file, the loaded playlist varies. This enables sequence of the M3U8 file's dynamic adjust and storage in database every time when a M3U8 file is requested by the client end.

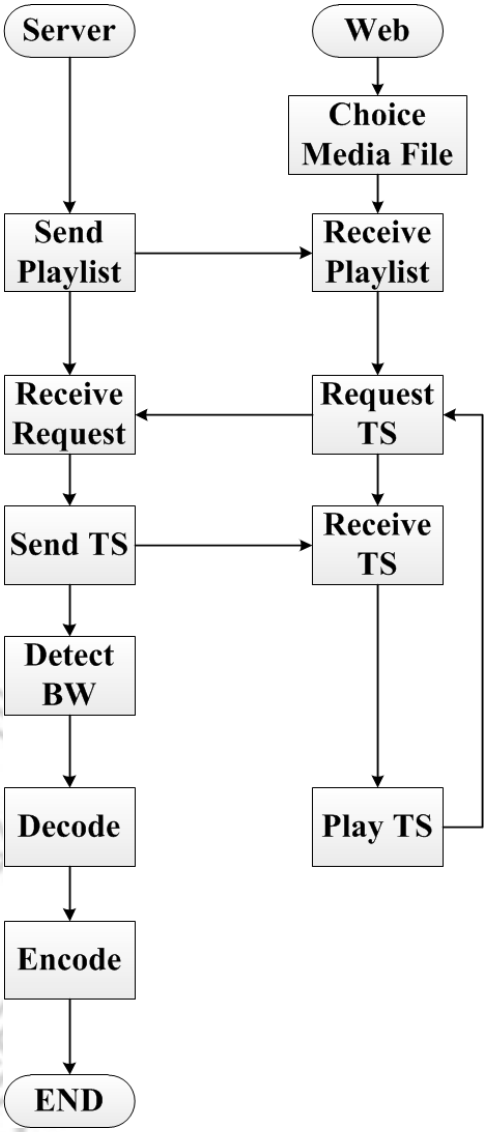

Fig. 8: Flowchar(first version)

```
<html>
<body>
<video src="http://ip/hls.m3u8" controls autoplay>
</video>
</body>
</html>
```

Fig. 9: Standard M3U8 playlists of HLS

The SEQUENE number indicates streaming file ID being played at the client end and enables it to check for new streaming files for download and playback. The contents of the M3U8 file replied to each connected client end varies from each other with its name varies with the client end's static IP.

```
<html>
<body>
<video src="http://genm3u8.php" controls autoplay>
</video>
</body>
</html>
```

Fig. 10: The code of dynamic generate M3U8

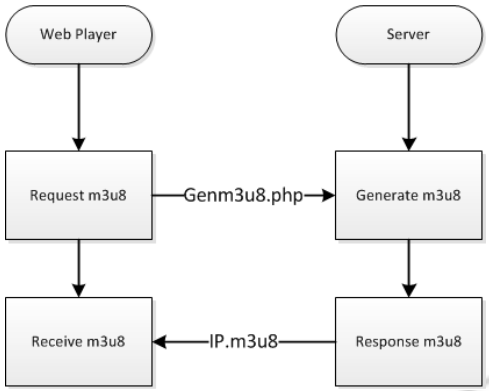

Fig. 11: The flowchart of dynamic generate M3U8

This makes instant playlist generation by PHP possible. When the client end requests for a playlist through HTML5, it runs a PHP program to generate the playlist and send it to the client end as shown in Figure 10 and Figure 11. It illustrates the M3U8 file's dynamic generation process.

After the M3U8 playlist is received at the client end, it plays the streaming file set by it and interfaces with the client end in PHP. When the client end requests to get a streaming file that can be downloaded, it runs the streaming file retrieval PHP once. Here the PHP downloads a playable streaming file from the server end, sends it to the client end, and measures the time required for transmission to check bandwidth changes between the client end and server end. It then uses the new bandwidth variance to get the new bit rate for next streaming file with formula 1. The server end then starts encoding next streaming file as the cached online streaming file for next polling as shown in Figure 8.

C. Dynamic transcoding

This paper practices a instant dynamic mechanism to encode a H.264 based MPEG4-TS file through FFmpeg, slice it into desired video clips in a fast pace, and delivers it to HLS for online streaming playback. FFmpeg features scores of parameters. This program employs its dynamic encoding parameter to convert images into x264 format based TS and sounds into mp3 format based file as well as to specify bit rate of image output. The FFmpeg dynamically adjusts relevant parameters of H.264 including images' chroma, group of picture (GOP) settings and sizes of DCT matrix. With the "-async" parameter it forces every streaming file to synchronize its audio and video file in the beginning to ensure concurrent A/V encoding and playback for the best viewing experiences.

```
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="Chinese
     ",AUTOSELECT=YES,FORCED=YES,URI="gensubm3u8.php
```

Fig. 12: The tag of HLS subtitles

```
printf("#EXTM3U\n");
printf("#EXT-X-MEDIA-SEQUENCE:%d\n", $seq);
printf("#EXT-X-TARGETDURATION:30\n");
printf("#EXTINF:30,\n");
printf("subtitles.webvtt\n");
```

Fig. 13: Dynamic generate subtitle

D. Video subtitle

For streaming file shot in a foreign language, viewers may lack the required language capability for full comprehension and may end up with poor viewing experience. Video streaming that supports subtitle function not only can provide translation texts but also can enable the streaming service provider to add some extra text information including subtitle ads. and banners. The practice of this paper employs WebVTT technology. The WebVTT technology is designed by W3C to provide subtitle for HTML5 streaming. This paper will revise the W3C specifications to comply with the proposed structure.

The output format of the M3U8 file needs be modified to provide subtitle function by changing the PHP program for M3U8 file generation with added subtitle tag as shown in Figure 12. HLS's #EXT-X-MEDIA:TYPE=SUBTITLES tag is used for set up and provide subtitle function; GROUP-ID="subs" is used to pair with corresponding streaming file; NAME="" can be set to display in player program for subtitle selection; AUTOSELECT=YES is used for auto selection ON or OFF; FORCED=NO is used for subtitle load default On or OFF; and URI="subtitles.m3u8" is used for selecting M3U8 file for mounting.

The subtitle practice modifies the contents in the URI tag from mounting the WebVTT subtitle to mounting the PHP program that auto generates subtitle M3U8 file paired with the playback streaming as shown in Figure 13. With this modification, the streaming file then reads in the paired subtitle file (WebVTT file). The program retrieves length of the subtitle's timeline to set up the DURATION parameter dynamically for single subtitle file mounting. Standard WebVTT requires individual subtitle file for each streaming video to enable streaming provider refresh subtitle during playback. This provides an easier approach while reducing the complexity in subtitle file mounting.

E. Bandwidth detection

Bandwidth measuring is critical. Precise bandwidth measuring between the client and server ends may improve quality

TABLE III: The table of media_list

item	type	description
Name	CHAR	Record video file name
Duration	BIGINT	Record video length(second)
BitRate	BIGINT	Record video bitrate(kbps)

of dynamic encoded videos and make the most of available network bandwidth. This paper provides three bandwidth measurement methods.

1) Provide speed measurement program for download at web page interface in JavaScript. Download a test file from the server end to measure current bandwidth. There is an issue with this method. The measuring activity consumes existing bandwidth that cuts bandwidth available to online streaming, worsens playback fluency, and does not measure the actual bandwidth variation between the client and server end accurately.
2) Revise program at web server to build up a socket in the server to measure transmission time during video streaming for bandwidth variation calculation. There is an issue with this method. You don't have the flexibility in selecting a proper web server when setting up the server nor the capacity to revise a program found in every market available web server.
3) The best method in solving these issues is to code a PHP program as the download interface between the client and server end. The client end requests a download by running a PHP program for easy download as required by the client end and easy measuring download time for bandwidth variation calculation. This helps in provide design requirements.

F. HLS server

The HLS is designed to set up an online streaming system in fast pace. Once the web server is set up, you can adjust server's MIME settings to support extensions required by HLS. To set up a Ubuntu platform with Apache web server installation, you need to modify "/etc/apache2/mods-enabled/mime.conf" for couple of file format supports.

G. Database design

This program requires a database system to store program operation data as the reference in completing every required function. This program set up a MySQL database with two tables named "media_list" and "session".

The "media_list" table stores list of files that can be played by the server end. This table contains fields of file name (Name), lasting time (Duration) and original video's bit rate (BitRate) as shown in Table III. Once a web interface is opened, see Figure 14, it retrieves a list of playable videos in the database for users' selection. A video query function is provided in the page. For a file list too long to display in a limited number of pages, viewers may find their desired video by this query function.

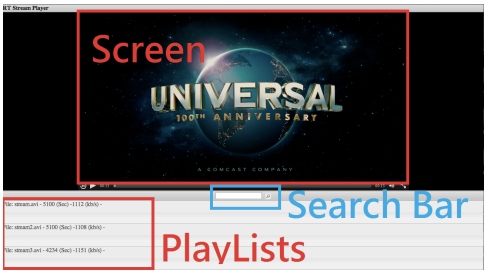

Fig. 14: Client interface

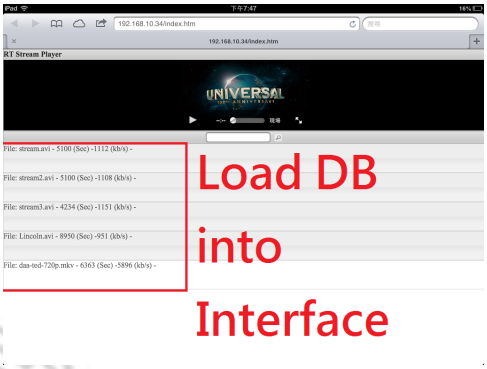

Fig. 15: Loading database into interface

The "session" table keeps current playing status of each user. Its "address' field maintains users' IP address to separate their storage data. The "bandwidth" field keeps measured bandwidth variation of each user that is used for set up the conversion parameter for next streaming file encoding. The "file" field keeps name of the file users selected for viewing. It is also the value of the file parameter for video encoding. The "sequence" field keeps users' current playback status. When used together with the "file" field it resumes the playing from where it is stopped by any interruption, e.g., user closed the program unexpectedly.

H. Interface design

This section codes a Web UI with jQuery. The latter is a commercial JavaScript library for easy and fast function designing. It retrieves data of files that can be played from the database to display in the playlist including name, original bit rate, and duration of the video. This Web UI enables users to operate web based player and query the database for videos available at the server with the same interfaces in mobile devices and computers as shown in Figure 15.

V. PERFORMANCE ANALYSIS

This paper is aimed at following goals before designing the AHLS:

1) Enables users to play streaming file in browser without installing any plug-ins for simplified streaming playback operation. Early streaming playback requires server end

TABLE IV: The different of streaming technology

Streaming	Subtitle	Environment	Plug-in
Adobe HDS	No Supported	IE, Safari, Chrome	Flash
MS Smooth	No Supported	IE, Safari, Chrome	SliverLight
DASH	No Supported	IE, Safari, Chrome	DASH VLC plugin
Apple HLS	Only Support VOD	iOS, Safari, Android	No need
mp4	Supported	All devices	No need
WebM	Supported	Chrome, Android 2.3.3 and up	No need
AHLS	Supported	iOS, Mac Safari, Android 4.0 and up	No need

coding and transmission protocol specific players to play streamed file, e.g. RTP/RTSP protocol.

2) Users may play video streaming with mobile or computer devices for streaming file viewing as long as a network connection can be established.
3) Enable users to play fluent video streaming in any networking bandwidth including landline, mobile and wireless environment. Conventional streaming service transmits files of the same quality that may impedes fluent streaming playing when lack of bandwidth. In cases like this, the player keeps on buffering streaming files until it can play the entire file fluently. This leads to poor viewing experience.
4) Enable mounting streaming in video streaming to provide translation texts for the video and other instant text information.

Table IV Reviews subtitle, playing platform, and browser with plug-in. With respect to browser that can mount WebVTT subtitles, the W3C now supports HTTP progressive download protocol and so mp4 and WebM may mount subtitles through HTML5 successfully. As HLS and AHLS is concerned, both support WebVTT subtitles on iOS platform. This enables the latter to mount subtitles successfully. HLS supports subtitles for VOD only while AHLS for instant streaming. With respect to plug-ins, current self-adaptive and progressive streaming technology mandates plug-ins for playback with a browser except HLS and AHLS. The latter two fully support streaming file playing in a browser with HTML5 support. In progressive download technologies, both mp4 and WebM can play streaming files with HTML5.

Table V and Table VI review performance of HLS and AHLS. The HLS technology is designed to start downloading next streaming slice when playing last slice. For a streaming slice of length in 10 seconds, the browser starts downloading the next slice when the playing of last slice has lasted for 5 seconds. Table V indicates that HLS's fixed bit rate design results in streaming slices in fixed size. When bandwidth is getting smaller it requires more time to transmit a file of the same size. For a bandwidth of 0.5Mbps, the streaming playback stops and the transmission cannot complete in 5 seconds. This hampers users viewing experience as the playback becomes broken. Table VI indicates that AHLS's design enables it to adjust next streaming slice's bit ratio according to current bandwidth variation adaptively. When bandwidth is getting smaller, the AHLS changes its bit rate accordingly to reduce streaming slice's size for shorter transmission time. In a test environment, all streaming slice files can be downloaded in a limited time

TABLE V: HLS performance

Bandwidth(Mbps)	2	1	0.5	0.25
Slice file size(Mbytes)	4.1	4.1	4.1	4.1
Transmit time()	2.05	4.1	8.2	16.4

TABLE VI: AHLS performance

Bandwidth(Mbps)	2	1	0.5	0.25
Slice file size(Mbytes)	5.688	3.535	1.247	0.803
Transmit time()	2.844	3.535	2.493	3.229

for fluent streaming playback and better viewer experiences. Fluent streaming playback at the expense of poorer image quality is acceptable. The AHLS design fits full range of bandwidth variation to relieve users from worrying about their playback environment.

VI. Conclusion

This paper is aimed at adjusting video bit rate dynamically according to existing bandwidth and providing online streaming video at the optimum image quality with limited image encoding parameters. The following conclusions are reached from studies described herein.

1) The test results indicates that the bit rate of video can be adjusted dynamically according to changing bandwidth between the client end and server end to enable users' smooth playback of online streaming in any networking environment including landline, wireless and mobile.

2) The practical web operation interface can work with iPhone, iPad, Android 4.0 mobile phone and Mac computer to provide the same operation interface for fluent playback. It plays online streaming without a client end program. It also plays in a browser without any plug-in like Adobe's Flash.

3) It may extend its application to create mobile theater environment by connecting tablet computer to TV or projector.

Making the most of known existing bandwidth by adjusting bit rate effectively is a critical task. Current formulas can deal with most bandwidth variation scenarios. It may be better to have a mechanism to estimate changes in bandwidth so that self adaptive bit rate for video encoding can be available before the bandwidth is known. This not only frees the system from waiting for bandwidth detection but also helps in getting closer to instant encoding synchronized with video playing. The result is more efficient bit rate adjustment and immune from impacts of fluctuating bandwidth variation environment.

As only HLS playback is supported now, available playback platforms are limited. The program in this paper has taken cross platform versatility into account. It is expected to add

more platform supports in future for fully cross platforms online streaming system.

REFERENCES

[1] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha, "Streaming video over the internet: approaches and directions," in *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 11, 2001, pp. 282–300.

[2] Report: Video accounts for half of all mobile traffic; android biggest for mo- bile ads. [Online]. Available: http://tinyurl.com/82svkle

[3] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann, "Dynamic adaptive http streaming of live content," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, 2011, pp. 1–8.

[4] C. Y. Hung, "Live broadcast and vod system based on http live streaming," , 2012.

[5] S. K. Hsieh, "Research and design of video adaption algorithm based on http live streaming," , 2012.

[6] Webvtt: The web video text tracks format. [Online]. Available: http://dev.w3.org/html5/webvtt/

[7] Video subtitling and webvtt. [Online]. Available: http://html5doctor.com/video-subtitling-and-webvtt/

[8] Http live streaming examples. [Online]. Available: https://developer.apple.com/resources/http-streaming/examples/

[9] Apple http live streaming. [Online]. Available: https://developer.apple.com/resources/http-streaming/

[10] K. J. Ma and R. Bartos, "Http live streaming bandwidth management using intelligent segment selection," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, 2011, pp. 1–5.

[11] T. Kupka, P. Halvorsen, and C. Griwodz, "An evaluation of live adaptive http segment streaming request strategies," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, 2011, pp. 604–612.