MPEG-DASH-Compliant Client-Server-Based Video Streaming

Project of Systems Support for Continuous Media

HU Sixing[1] CHI Ji[2] YU Xiaolong[2]

[1] Department of Computer Science, [2] Department of Electrical and Computer Engineering
National University of Singapore

1. Abstract

Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, is an adaptive bitrate streaming techniques that enables high quality streaming of media content over the Internet delivered from conventional HTTP web servers. MPEG-DASH works by breaking the content into a sequence of small HTTP-based file segments, each segment containing a short interval of playback time of content that is potentially many hours in duration, such as a movie or the live broadcast of a sports event.

MPEG DASH is a developing ISO Standard (ISO/IEC 23009-1) that should be finalized by early 2012. As the name suggests, DASH is a standard for adaptive streaming over HTTP that has the potential to replace existing proprietary technologies like Microsoft Smooth Streaming, Adobe Dynamic Streaming, and Apple HTTP Live Streaming. A unified standard would be a boon to content publishers, who could produce one set of files that play on all DASH-compatible devices.

In this paper, we aim to creating a DASH-compliant live streaming system. In order to implement this, firstly, we capture a video on an ASUS Transformer and store it as an MP4 files, then program on the Android in Java with the Android Studio IDE to split the MP4 file into streamlets and upload it to a web server. On the web server side, we create scripts in PHP language to create a playlist. Besides, we also implement a simple Android DASH media player to play these video.

2. Introduction

The system is separated into two parts - Client and Server. First we capture a video on an ASUS Transformer and store it as an MP4 files. In our program we use use *MediaRecorder* in Android to record and save video. When a video is chosen by user, a new thread starts to do the segmentation and upload these streamlets. For LIVE model video capture, it would be different to VoD in implementation since mp4 video will only be created until recording stop if we use *MediaRecorder* to capture video. In live mode, it is still used to record video. For every 3 seconds, the program stops recording and a mp4 of three-second video is created. Then recorder restarts to record another 3 second video.

On the server side, the main function is to receive and transcode video segments, and then generate corresponding MPD as well as Playlist. When the file *upload.php* is requested by the client, the server will receive the uploading packages and transcode them into 3 resolutions. Then the server will generate a MPD file using an XML format file. To realize the live model, the value of endIndex can be set as a negative value to indicate the player that this is a live video. And by using the header information which client sends, this kind of MPD for live video can be even generated before the first segment uploading so that satisfy the real-time requirements.

3. Summary of existing streaming techniques

Streaming media usage has grown exponentially over the past years, both for entertainment purpose and as vehicle for organizations to market, sell, and support their products and services, as well as for internal communications and training. For many such organizations, streaming video has transitioned from a ‘’nice to have’’ curiosity to a mission critical technology.

At the same time, streaming technologies have transitioned dramatically since the early years. Streaming technologies like Adobe Flash, Apple QuickTime, and Microsoft Windows Media and Silverlight all include certain common components in their solutions. These include a player to play the media on the viewer's computer or mobile device, a defined file format or formats that the player will play, and often a server component that offers features like digital rights management and live streaming.

Adaptive bitrate streaming was introduced by Move Networks and is now being developed and utilized by Adobe Systems, Apple, Microsoft and Octoshap.

These are some implementations of adaptive bitrate streaming:

MPEG-DASH is the only adaptive bit-rate HTTP-based streaming solution that is an international standard. MPEG-DASH technology was developed under MPEG. Work on DASH started in 2010; it became a Draft International Standard in January 2011, and an International Standard in November 2011. The MPEG-DASH standard was published as ISO/IEC 23009-1:2012 in April, 2012.

Dynamic streaming is the process of efficiently delivering streaming video to users by dynamically switching among different streams of varying quality and size during playback. This provides users with the best possible viewing experience their bandwidth and local computer hardware (CPU) can support. Another major goal of dynamic streaming is to make this process smooth and seamless to users, so that if up-scaling or down-scaling the quality of the stream is necessary, it is a smooth and nearly unnoticeable switch without disrupting the continuous playback.

HTTP Live Streaming (HLS) is an HTTP-based media streaming communications protocol implemented by Apple Inc. as part of QuickTime X and iOS. HLS supports both live and Video on demand content. It works by breaking down streams or video assets into several small MPEG2-TS files (video chunks) of varying bit rates and set duration using a stream or file segmentation. One such segmentation implementation is provided by Apple. The segmentation is also responsible for producing a set of index files in the M3U8 format which acts as a playlist file for the video chunks. Each playlist pertains to a given bitrate level, and contains the relative or absolute URLs to the chunks with the relevant bitrate. The client is then responsible for requesting the appropriate playlist depending on the available bandwidth.

Smooth Streaming is an IIS Media Services extension that enables adaptive streaming of media to clients over HTTP. The format specification is based on the ISO base media file format and standardized by Microsoft as the Protected Interoperable File Format

QuavStreams Adaptive Streaming is a multimedia streaming technology developed by Quavlive. The streaming server is an HTTP server that has multiple versions of each video, encoded at different bitrates and resolutions. The server delivers the encoded video/audio frames switching from one level to another, according to the current available bandwidth. The control is entirely server-based, so the client does not need special additional features. The streaming control employs feedback control theory. Currently, QuavStreams supports H.264/MP3 codecs mixed into the FLV container and VP8/Vorbis codecs mixed into the WEBM container.

upLynk delivers HD Adaptive Streaming to multiple platforms, including iOS, Android, Windows 8, Roku and all PC/Mac/Linux browser combination by encoding video on the cloud using a single non-proprietary adaptive streaming format. Rather than streaming and storing multiple formats for different platforms and devices, upLynk stores and streams only one.

4. Description of Application Implementation

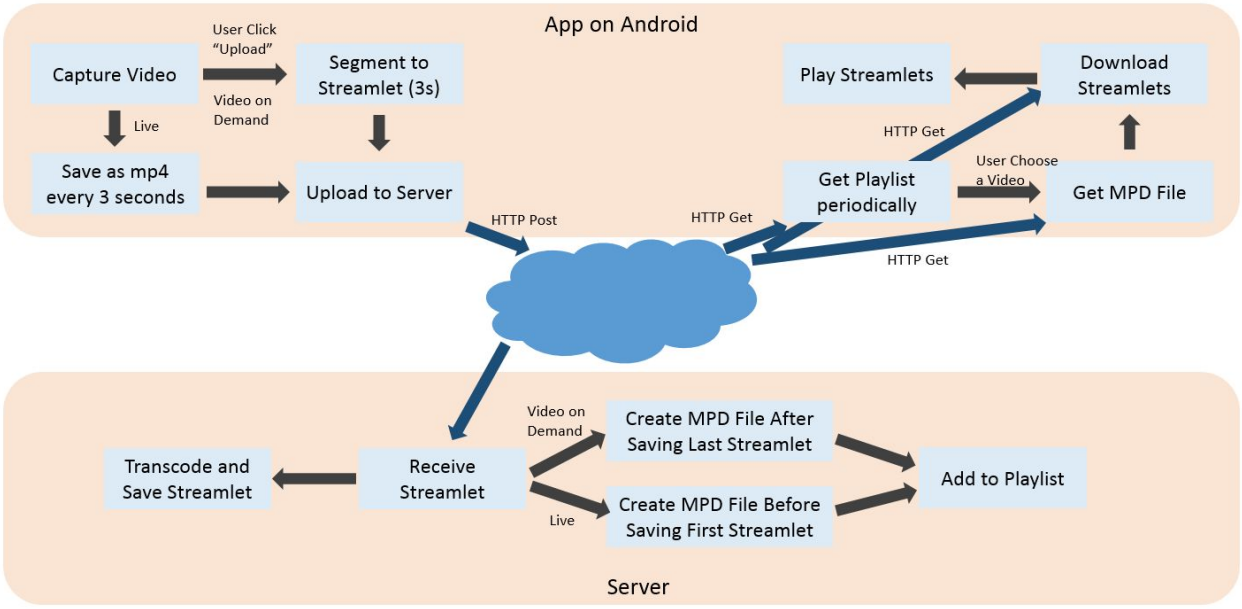

Fig 4-1. Framework of DASH Client-Server-Based video streaming system

The system is separated into two parts - Client and Server. Receiving Streamlets of video, transcoding streamlet into different resolution and into iOS playing format, creating MPD and playlist are finished on server side. Video capture, segmentation and playing run on client side. Logically, client can be divided into two parts - Video capture and Video player. For video on demand (VoD) mode, mobile video capture will record whole video and save in local storage. When user chooses a video to upload, the app will segment this video and upload it as soon as segmentation finished. For live mode, mobile video capture will save every three second video to mp4 file during recording. As soon as a mp4 file is generated, it will be uploaded to the server. There is very little difference between VoD and Live mode to mobile player. App refreshes playlist periodically to always get newest video list. When user chooses a video item, an MPD will be downloaded first and all streamlets are downloaded according to the MPD. Once there are enough streamlets in buffer, the video player will start play the video streamlet by streamlet. The framework of our system is illustrated in Fig 4-1.

4.1 APP: Video Capture and Uploading

4.1.1 Video Capture and Uploading for VoD Mode

Our program using *MediaRecorder* in Android to record and save video. When a user click the recording button, MediaRecorder begins initialization and starts recording. When a user click the recording button again (means stop for this time), MediaRecorder stops recording and full mp4 file is created and stored in local storage.

Users have two ways to upload their videos. One is to upload the video just recorded immediately. Another one is to upload later when they want to, which allow users to upload any videos at any time. When a video is chosen by user, a new thread starts to do the segmentation and upload these streamlets. Third party library - *MP4Parser* are utilized to do segmentation to the mp4 video, *httpclientandroidlib* are utilized to do network communication including Http Post.

Segmentation will be done first if the video has never been segmented. After reading the video, scan the video once to find all the synchronization points on which mp4 video can only be cut to separate self-contained streamlets. For every 3 second position, a nearest synchronization point is used to segment the video to a series of 3 second short videos. These streamlets are also stored in local storage. The uploading starts as soon as all streamlets are saved. Except for the streamlet, video name, segment ID and segment number are also packaged together to upload to server by HTTP Post. The next package will be uploaded only if the previous package has uploaded successfully. If the network connection is interrupted, the uploading will stop and will restart when user ask to upload again. Our system can guarantee that one package will not be uploaded twice. It will be introduced in 4.4.2 part in details.

4.1.2 Video Capture and Uploading for LIVE Mode

Live mode is different to VoD in implementation since mp4 video will only be created until recording stop if we use *MediaRecorder* to capture video. *MediaRecorder* is a very convenient API and in live mode, it is still used to record video. For every 3 seconds, the program stops recording and a mp4 of three-second video is created. Then recorder restarts to record another 3 second video. There are some problems using this method and we will discuss it in 4.5 part. Once a mp4 streamlet is saved, the uploader in another thread will upload it to server immediately with same information described in 4.1.1 part except the segment number. For live mode, the segment number is set to negative integer since the program does not know the number of segments. A tag indicating live mode is also packaged to inform server that it is a segment for live mode.

4.2 Server: Receiving, Transcoding and Generating Playlist and MPD

On the server side, the main function is to receive and transcode video segments, and then generate corresponding MPD as well as Playlist.

The file named *data.txt* is generated by server to record the information of a certain video's segments number. Before sending segment, the client will send a header (package) to the server to indicate which video is going to be sent. And the server will search this video's *data.txt* accordingly and reply the client with the information of the existence/IDs of the segments which belong to the above mentioned video. The client won't upload any exited segment to the server by analyzing the returned strings. And by this method, a resumed upload after an interruption of network connection can also be realized.

When the file *upload.php* is requested by the client, the server will receive the uploading packages and transcode them to .mp4 and .ts files in three resolutions using “mp42ts” and “convert.sh” tools.

The client will inform the server that "I'm Done" by requesting executing *process.php* when all the segments are successfully uploaded, and then the server will generate a MPD file using an XML format file, which will be added to the playlist afterwards. Especially, to remove the duplicates in the playlist, checking whether a same MPD file exists will be done before adding any MPD to the playlist.

In terms of MPD, it will be generated with dynamic url, e.g.:

```
...
<UrlTemplate indexURL="$$.mp4" sourceUrl="$videoname/720x480/$filename---"
endIndex="5">
</UrlTemplate>
…
```

The client (player) will add the indexURL to the tail of sourceUrl, and replace the parameter "$$" with a figure which starts from 1 to the value of endIndex, so the file paths for client are:

```
.../$videoname/720x480/$filename---1.mp4
.../$videoname/720x480/$filename---2.mp4
.../$videoname/720x480/$filename---3.mp4
.../$videoname/720x480/$filename---4.mp4
.../$videoname/720x480/$filename---5.mp4
```

To realize the function of playable live videos, there is no time waiting for all the segments being uploaded, in other words, neither the server nor the recorder knows the sum of segments of a

live video. Hence the value of endIndex can be set as a negative value to indicate the player that this is a live video. And by using the header information which client sends, this kind of MPD for live video can be even generated before the first segment uploading so that satisfy the real-time requirements.

4.3 APP: Video Player and Downloading

No matter the mode is VoD or Live, the video player's part of app is almost same. Third party library - *httpclientandroidlib* are utilized to do network communication including Http Get. A xml format file of playlist which saves all MPD files URL of all the videos on server are downloaded every five seconds. The content in playlist will be shown on the screen and users will choose a video they want to watch. When a video is chosen, the app will first download its MPD which includes base url of the video, url template of the video in different resolution, first index and end index of the video in different resolution. If it is a live video, the end index is negative and program will set the end index as infinity.

Activity thread is in charge of play video and downloading thread is in charge of download streamlets according to information from its MPD. A shared buffer is used by these two thread. The download speed, video quality and play procedure are controlled by buffer management system which is introduced in 4.4.1 part in details. Once a streamlet is downloaded, it is inserted to shared buffer. When to start download next streamlet and what the quality of next streamlet are determined by buffer management system. When there are enough streamlets in buffer, the buffer management system will inform playing activity to start or restart playing video. For video player, if one streamlet finishes, the player will immediately get next streamlet and continue to play next streamlet. If it is the end of video, the player will close downloading thread and return to main activity where users can choose another video through playlist. If it is not the end of video but the buffer is empty, the player will wait until buffer management system informs it that the buffer is ready or to stop playing since maybe the network is down. If the download times for a streamlet exceed a certain threshold, the sign of network disconnection is reported.

4.4 Features and functionalities

4.4.1 Buffer Management System

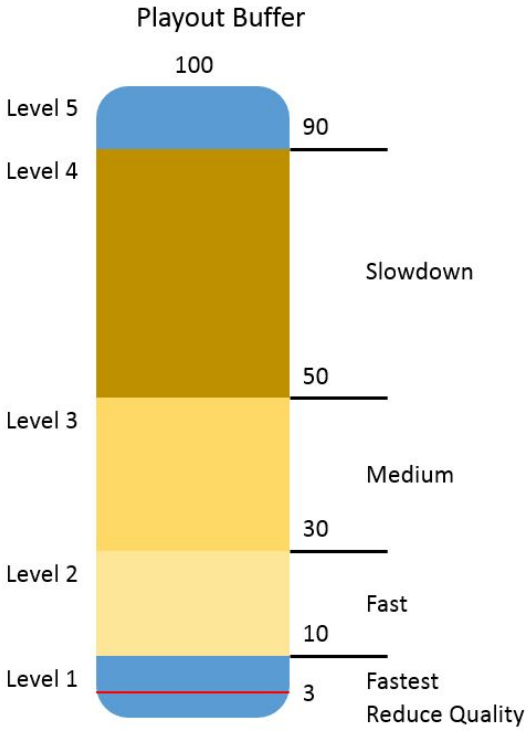

Fig 4-2. Buffer management structure

The buffer management system includes a shared buffer read and written by player activity thread and downloading thread and a management system. In our program, the size of shared buffer is 100 which can at most save about 5 minutes video. The most important part of our buffer management system is the management system.

The management system has a series of strategies to control download speed, streamlet resolution and control when to start playing video. As shown in Fig 4-2, the shared buffer is separated into 5 parts and the object of control is to maintain the usage of buffer around 50% and try best to guarantee 10% usage. When the buffer usage is under 10%, the interval time of downloading each streamlet is 0. For the other four usage from level 2 to level 5, the interval time of downloading each streamlet is from 1 second to 4 seconds. The red line in Fig 4-2 is the alarm line. If the usage of buffer is less than 3%, the resolution of streamlet need to be downloaded decrease until the usage exceed 3%. If the usage keeps larger than 3%, the resolution of streamlet will increase. Besides, a threshold used to start and restart video is set according to the playout mode and network condition. When the usage of buffer exceed this

threshold, the buffer management system will inform the player and the player will start to get streamlet from buffer and to play.

4.4.2 Resumed Uploading Support

Each video uploaded to server has a unique ID which guarantee that different video will be stored on different place on server and same video must be stored on same place on server. Our program promises that same segmented video will not be uploaded twice. When the network interrupts and restores later, the app will only upload rest of streamlets. To avoid uploading duplicated streamlets, when the client start uploading a VoD, it first send a header to server to inform the video ID of the upcoming video and get a response from server including all segments' ID which corresponding segments already exist on server. Then the client will check and only uploads rest of segments.

4.4.3 Playing Live video from most recent point

For live video, the start index is not default 1, but is ID of the segment which is received by server most recently. This simple modification allows video player to start play at most recent point and get best synchronization.

4.5 Challenges and Difficulties

The difficulty we encountered during our development is Live streaming. We meet two difficulties. The first one is segmented video and recording video at the same time. What we use to record video is Android API *MediaRecorder* which can only create mp4 format file after recording finish. It is not suitable for live streaming. The second difficulty is relatively long time to do transcoding. To realize live streaming, all operation to one segment must be done in limited time slot that is the length of one streamlet, in our case is about 3 seconds. When we discuss with other teams, they told that even the time of transcoding is about 3 seconds. When we finished our project, we test that the whole procedure from uploading to downloading is about 3 seconds. Fortunately our program can handle this problem when network is perfect, but what if the network is not perfect?

For the first difficulty, we simply record 3 seconds and stop to create a mp4 streamlet and then restart recording and stop again in 3 seconds. This cycle repeats until user stops recording. But there is a problem of this method. Since the start and stop procedure is slow, every cycle there are nearly one-second frames loss. A lower API need to be used to solve these problem. The video stream need to be obtained at any time and there should be a mp4 parser to create mp4 format video from media streams. That is very complicated and we did not apply in our project.

For the second difficult, we delete all the function that is not necessary for live video to accelerate operation. Resumed upload is not supported in live video, response information is reduced to lowest level. But the speed is still not enough is the network environment is not good. Transcoding takes most of time and it is hard to reduce the time to transcode. Therefore it is very hard to solve this time problem.

5. Results and Discussion

5.1 Achievement

We finish a client-server system based on DASH which supports video on demand and live video. Both recorder and player are on the one android app so user do not need to install two app to use our service. For the client, it can record a video and upload as live video or as VoD video. User can upload any video at any time. User can choose a video to watch from playlist. The system support VoD video. Client can record the video and segment the video on-the-fly, can upload segmented videos, can download playlist, MPD and streamlets, can play video by concatenating all streamlets. The server can receive and send streamlets according to request from client, can transcode streamlets to lower resolution and to iOS based format, can create MPD, M3U8 xml file and playlist xml file. Besides, our system supports resuming a failed upload from the same point where the network error occurred, supports buffer management to control download bitrate, supports playing two videos on two different mobile client simultaneously. An apple browser can also see the video from our index webpage.

5.2 Experiment

We test recording, segmentation, uploading, downloading and playing for video on demand and live video. For VoD, there is no problem to record, segment, upload and download. We test playing the video and it turns out that although we simply play video streamlet by streamlet, the whole video looks seamless. For live video, using our start-stop-restart-stop method, there will be 10 to 20 frames loss for every segment (about 3 seconds). The delay for live video is about 10 seconds since the video player buffer 3 streamlets before it begin playing video.

5.3 Discussion and Further improvement

The least latency of Dash video streaming is the length of one segment which is very large, thus Dash is not suitable for interactive media like video conference. The operation time for one segment is very long (We explain it in 4.5 part) and maybe longer than the length of one segment if the performance of server is not good or if the network is bad, thus Dash is also not a good choice for live video. The best application is video on demand. Since HTTP is easy to use in the program and is less influenced by network, DASH is a better choice to send and receive media over internet. When we do the program of our project, we find that it is easier to use DASH to send and receive video than other method we used before.

There are some things we can do further to improve our system and service. We can use lower API to manipulate recorder and camera more directly to get mp4 video during recording period to avoid any loss of frame. It will make our system provide real live video service. We also need to add a progress bar to let user control and watch the video again easily. After we finish those parts, our project will become a real product that can be used in real life.

6. Lowering end-to-end delays for MPEG-DASH streaming

End-to-end delay in a video streaming is primarily based on the stream buffers. The lower the buffers, the lower is the delay. Buffers can happen on all parts of a stream, encoder, streamer, server, CDN and player. In order to minimize the delays for MPEG-DASH streaming, there are some common research directions:

1) Encoder configuration: on Application/API level, nanoStream provides some functions to decrease end-to-end delay
2) Player configuration: The player buffers have a high impact on latency. Better choice on the player configuration can contribute to decreasing the delay.
3) Choosing appropriate segment size: The length of the segment have impact on the latency of DASH stream, thus choosing appropriate segment length can improve the latency.

7. Conclusions

We finish creating a DASH-compliant live streaming system, which supports both video on demand and live video. The client can record a video and upload it as live or video-on-demand. Then the server can transcode the streamlets to lower resolution and send it according to the request from the client. Then the user can watch the video anytime in the playlist. Besides, the server can also transcode the streamlets into IOS based form and send to the IOS user, but we have not yet implemented the live model part for the IOS and we will put some efforts on it for the further improvement for our system.

Finishing a project in a team is not an easy thing and think the difficult part of this project is to fit our server side php scripts with the client side codes, it took us tons of time to make 3 person's work into a whole one. Though function of our project is quite limited, we believe that after several improvements it can be a useful product in the real word.

8. Reference

[1] ISO/IEC CD 23001--6. 2010. Information technology -- MPEG systems technologies -- Part 6: Dynamic adaptive streaming over HTTP (DASH), Guangzhou, China, Oct. 2010.
[2] Stockhammer, Thomas. "Dynamic adaptive streaming over HTTP--: standards and design principles." *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011.
[3] httpclientandroidlib, Android library for HTTP: https://code.google.com/p/httpclientandroidlib/ [Accessed 17. November. 2015]
[4] Live Video Encoder-Low Latency Streaming
HTTP: http://www.nanocosmos.de/v4/documentation/live_video_encoder_-_low_latency
[5] Adaptive bitrate streaming
HTTP: https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming