# ADAPTIVE BIT-RATE STREAMING

## Minimizing End-User Buffer Times in Real-Time Video Delivery

**Ted D. Monchamp***
**Graduate Student, M. S. Program in Computer Science, Rivier University**

### Abstract

*This paper describes an overview of adaptive bit-rate streaming as a content delivery technique. General principles will be discussed as well as an introduction to several of the current leading adaptive streaming technologies and how they function. Adaptive streaming is a technique used for client-driven video streaming over HTTP, and aims to provide end users with higher-quality playback by adjusting the stream to fit the client's current network and processing conditions.*

### Introduction

In 2012, a reported 57% of all global Internet traffic was video. This figure is projected to reach 69% by 2017, not including peer-to-peer file sharing. Most of that traffic will be on mobile devices. [1] The additional network load results in much higher congestion, which would reduce the quality of service (QoS) for end users. Previously, video content delivery networks (CDN) would host videos and stream them at a constant bit-rate, regardless of network congestion. In the event that a user's downstream bandwidth could not stream enough chunks of a video to maintain continuous playback, the video would stop and the user would have to wait while enough chunks of the video download in order to resume playback, often with a now-familiar icon appearing in the foreground (see Fig. 1).

**Figure 1**. Sample Buffering Icon

Especially during peak usage hours in the evening, having to constantly pause a video to wait for the player's buffer to refill proved to be unacceptable for any content provider looking to compete against broadcast television and cable providers. Adaptive bit-rate streaming was developed as a technique to mitigate this issue. [2] The CDN provides several available bitrates for streaming which the client software can switch between seamlessly. Ideally, the difference between streams is gradual. This way, playback can occur without any noticeable interruptions for the consumer. [3]

### Predecessor Streaming Techniques

Traditionally, online video playback was handled by one of two major techniques: real-time streaming and progressive download. Streams would only contain a single file, and transmit segments to clients in the PLAY or RECORD states in real time relative to playback time on the server when the client is not



ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

in the PAUSE state. [4] This technique is susceptible to network congestion, where segments may be dropped if the client does not receive them quickly enough.

Progressive download sends segments of the video to the client where they are written to disk (usually in the browser cache). The download will continue until the entire file has been transferred to the client. The primary difference between this technique and a regular browser download is that playback can begin using the existing file segments before the entire download is complete. However, playback must pause if it catches up to the download of new segments and there is nothing left to play at that point. [5] [6]

## Overview of Adaptive Streaming Technique

Adaptive bit-rate streaming is a technique for video delivery using Hypertext Transport Protocol (HTTP). The technique, being HTTP-based, is inherently stateless. This means that the server hosting the video only knows of its current connections to client endpoints on the network. Each client must manage its own state. That is, the streaming client must determine for itself what the optimal stream characteristics should be based on local network and CPU load. [7]

The web server hosting a given video holds that video encoded in multiple different bitrates, each broken down into small segments that can be sent as requested to the client. [8] The client is provided with a manifest from the web server that contains information about the stream encodings available, and it requests the encoded stream at a bit-rate appropriate for local conditions (see Fig. 2). [3] The server maintains no record of what previous client request information was, and treats each HTTP request for a new segment individually. [9]

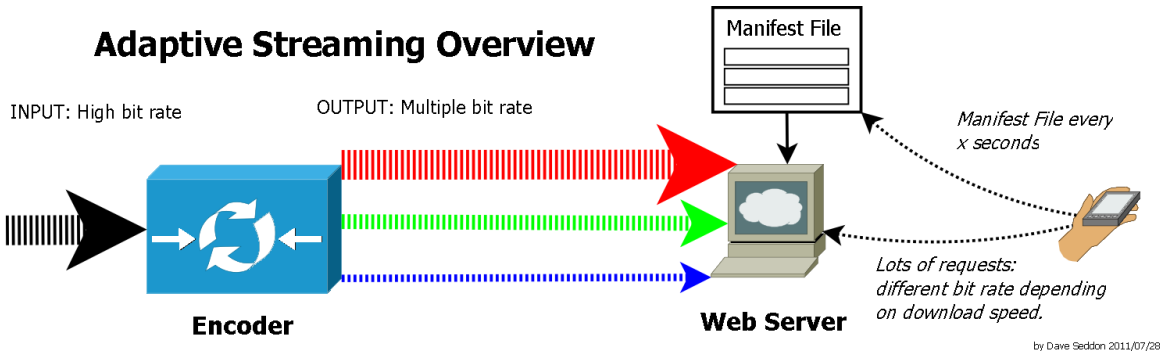


**Figure 2**. Overview of Adaptive Streaming Architecture (D. Seddon, Public Domain)

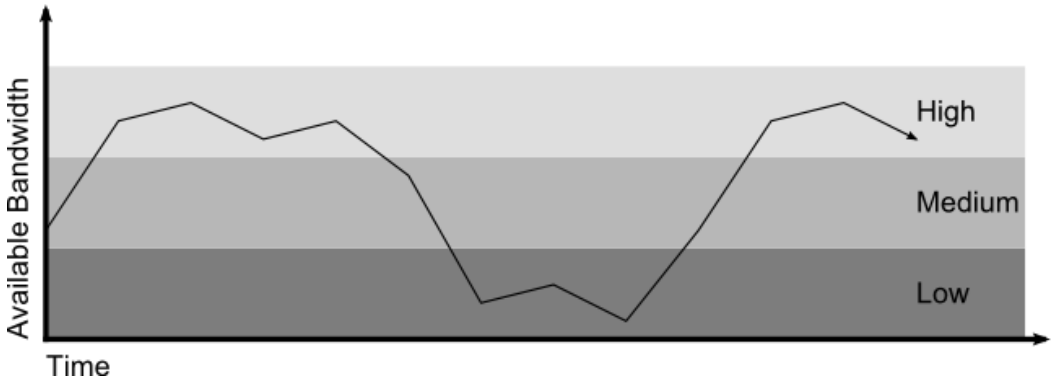


**Figure 3**. Stream Bit Rate as a Function of Network Congestion

The client will generally start with a segment of video at the lowest bit-rate. If the download speed of that segment indicates that more bandwidth is available and the CPU can handle playback, the subsequent segment is requested from the highest bit-rate afforded by the measured bandwidth. If bandwidth is observed at some point to decrease, and a segment download is taking too long, the client will request the next segment at a lower bit-rate in order to maintain real-time playback (see Fig. 3). In this manner, the video will only pause playback if the bandwidth decreases so much that the client cannot download the lowest bit-rate segments as often as they are needed. [7] [8]

## Benefits

### *Technical*

The foremost benefit of the use of adaptive bit-rate streaming is the stream quality for the end user. When properly implemented, though quality of the video stream may vary, the effect should be nearly unnoticeable, as opposed to having a break in playback from network congestion. There also won't be danger of firewall conflicts for the transmission with HTTP-based streaming as all traffic occurs on port 80, as opposed to an assortment of obscure UDP ports that are used for other streaming protocols. [5] [9] For the end user, the process of consuming adaptive video streams is seamless by design.

The content producers can also benefit from adaptive streaming. Extra encoding equipment is required initially to create the necessary streams in several bitrates, but ultimately the workflow is more streamlined, as the encoding and hosting software often work together to manage the videos automatically. Hosting of HTTP-based streams also doesn't require specialized hardware beyond regular web servers for serving content over the Internet. [9] [10]

### *Commercial*

Fragment URIs listed in the manifest do not have to refer to chunks of video in the current stream. Absolute URIs can be used to splice in segments from external CDNs. [11] These paths are often used for inserting advertisements into streams. As they can point to any script that returns a valid video segment, personalized ads can be added to the video content. The content provider will have to be aware though, that requests for video segments are made by the client and can be either altered or ignored. [12] The client video player should be designed accordingly so that such exceptions can be handled to maintain a balance between uninterrupted playback and ad revenue.

## Leading Implementations

### *Adobe Dynamic Streaming for Flash*

HDS is a technology designed jointly by Adobe and Akamai for use with Adobe's Flash Media Server. This adaptive streaming model is built to handle streaming using both HTTP and Adobe's pre-existing Real Time Messaging Protocol (RTMP), combining the features for progressive download and streaming. [13] This allows users to have instantaneous playback and the ability to seek to any point within the video. [10]

HDS streams videos packaged as either of two industry standards: MPEG4 (H.264/AAC) or Flash video (FLV). Both formats support multiple bit-rate streaming. When preparing a video for deployment on the web server, the file must be repackaged in the F4F format. These packages each contain a manifest to be sent to client machines indicating available stream formats (see Fig. 4), an index of the

included video fragments, and segments of the original source file which contain the URL-addressable video fragments. [10] [14] [15]

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns="http://ns.adobe.com/f4m/1.0">
   <id>myvideo</id>
   <duration>253</duration>
   <mimeType>video/x-flv</mimeType>
   <streamType>recorded</streamType>
   <baseURL>http://example.com"</baseURL>
   <drmAdditionalHeader url="http://mydrmserver.com/mydrmadditionalheader"/>
   <bootstrapInfo profile="named" url="/mybootstrapinfo"/>
   <media url="/myvideo/low" bitrate="408" width="640" height="480"/>
   <media url="/myvideo/medium" bitrate="908" width="800" height="600"/>
   <media url="/myvideo/high" bitrate="1708" width="1920" height="1080"/>
</manifest>
```

**Figure 4**. Sample F4F Manifest (OSMF Wiki) [14]

Client playback of HDS video streams requires the user to install only Adobe's Flash Player plug-in or their Air framework. Access to hosted video is handled by a custom application utilizing the Adobe/Akamai-designed Open Source Media Framework. The application can then request manifests from the server in order to retrieve the locations of the necessary video fragments for the stream (see Fig. 5). As users will generally have Flash installed already, minimal effort is required on their part, if any, in order to support Adobe HDS playback. [10]

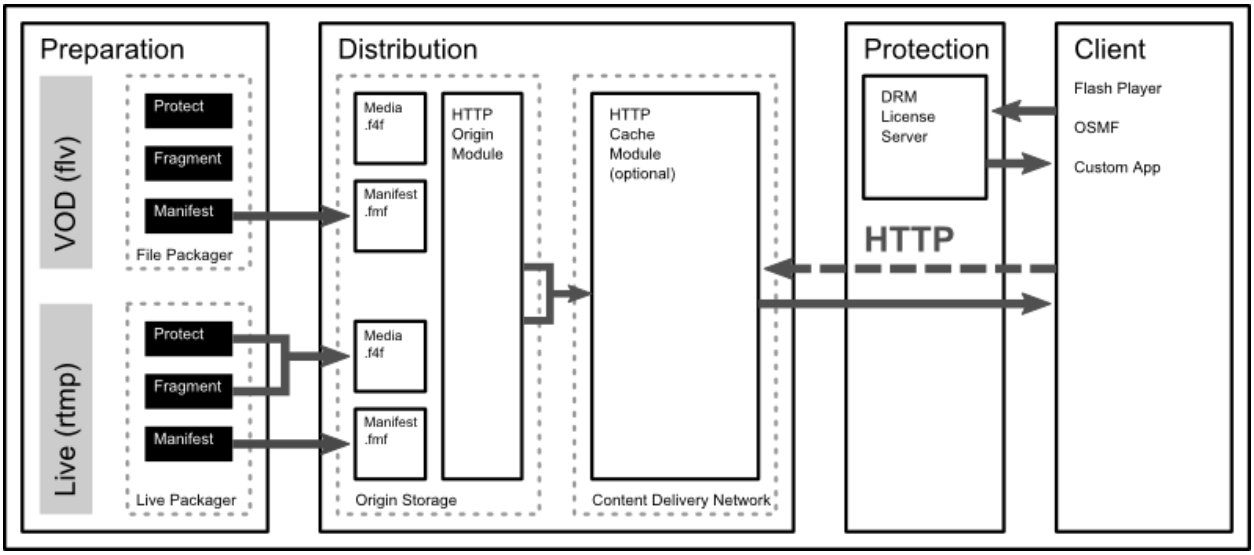


**Figure** 5. HTTP Dynamic Streaming Model

### *Apple HTTP Live Streaming*

HLS is an HTTP-based streaming protocol developed for use with iOS devices. The protocol is designed to enable video transmission over port 80, allowing more flexibility for mobile devices, which could attach to a variety of network and therefore won't have to deal with blocked obscure UDP ports. [16]

HLS operates by breaking a video stream into short chunks a few seconds in length. The server then maintains an extended M3U playlist that contains codec, bit-rate, and URI information about the video segments in the stream (see Fig. 6). The segments themselves must be valid MPEG-2 (DVD) or MPEG-4 (H.264/AAC) encoded files. When a client connects to the web server, it downloads a copy of the playlist for the desired video stream. Provided that the playlist parses correctly, the client then requests a segment of video at the appropriate bit-rate. Segments need not be downloaded in any particular order. Random access is allowed, so the client program is responsible for correct playback in any desired manner. [6] [16]

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:7794
#EXT-X-TARGETDURATION:15
#EXT-X-KEY:METHOD=AES-128,URI=https://priv.example.com/key.php?r=52s
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="English", \
DEFAULT=YES,AUTOSELECT=YES,LANGUAGE="en", \
URI="main/english-audio.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="Deutsch", \
DEFAULT=NO,AUTOSELECT=YES,LANGUAGE="de", \
URI="main/german-audio.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="Commentary", \
DEFAULT=NO,AUTOSELECT=NO,URI="commentary/audio-only.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=1280000,CODECS="mp4a.40.5",AUDIO="aac"
low/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2560000,CODECS="mp4a.40.5",AUDIO="aac"
mid/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=7680000,CODECS="mp4a.40.5",AUDIO="aac"
hi/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5",AUDIO="aac"
main/english-audio.m3u8
```

**Figure 6**. Extended M3U Master Playlist with Alternate Audio (E. Pantos) [16]

Throughout playback, the client must periodically reload the playlist unless the EXT-X-ENDLIST tag is present in the data. The web server can append data to the playlist, either with new segments, or URLs for other streams to be inserted into the current playback. This allows for the server to host live streaming content as well as video-on-demand. As the connection to clients is stateless, the server has no knowledge as to which clients require updates, so the clients must check the hosted playlist for any updates.

Digital rights management for HLS is handled using existing industrial standards. Encryption of streamed videos uses 128-bit AES with secret key sharing using HTTPS, along with either a device-based login or HTTP cookie. The necessary keys for decrypting a video for playback are provided in the playlist downloaded from the server. [16]

### *Microsoft Smooth Streaming*

Microsoft's Smooth Streaming technology was among the earliest implementations of adaptive streaming. Very primitive precursors to the present technique began in 1998 with NetShow Services 3.0, which allowed Windows Media Player to detect deteriorating bandwidth and reduce the frame rate of a video stream or, at worst, stream only audio. Between 2000 and 2003, the ASF wrapper was introduced with Windows Media Service 4.1, which allowed for multiple videos with varying bitrates to be

contained in the same file. Protocols developed at that time supported the ability of the player to switch bitrates during playback. However, a proprietary format was still required, and there was not yet any way of aligning the video streams or the audio in any timely manner, making switching between videos difficult, and certainly not seamless. Progressive download was not an option either, as only the server could manage the streaming sessions. [5]

The present iteration of Smooth Streaming was built as an extension of Internet Information Service (IIS) 7.0, Microsoft's web hosting platform. As with other adaptive streaming techniques, IIS Smooth Streaming is based on HTTP, so that the video hosting extension can be easily integrated with the rest of the hosted content on a web server. Web applications for consuming these video streams are developed using the Silverlight platform, which runs a subset of the .NET runtime designed for deploying applications to run entirely within the client browser. [5]

Videos encoded for streaming use the ISO standard MPEG-4 format as opposed to the original ASF package. The H.264 codec used for encoding the video itself has better compression and lower overhead for client parsing. The MP4 format is also supported across more platforms and already has built-in support for payload fragmentation (see Fig. 7). This means a single file can be hosted on-disk on the server that already contains the necessary segments to send to a client at any of several specified bitrates that have been encoded and stored within that file. Each fragment is also addressable by its timestamp instead of an arbitrary index. Creation of these files is also designed to be seamless for ease of implementation. Neither live nor video-on-demand encoding requires special dedicated hardware. The video feeds need only be run through MS Expression Encoder, which can handle the multiple bit-rate encoding, payload fragmenting, and deployment once the output stream parameters are configured. [5] [17]

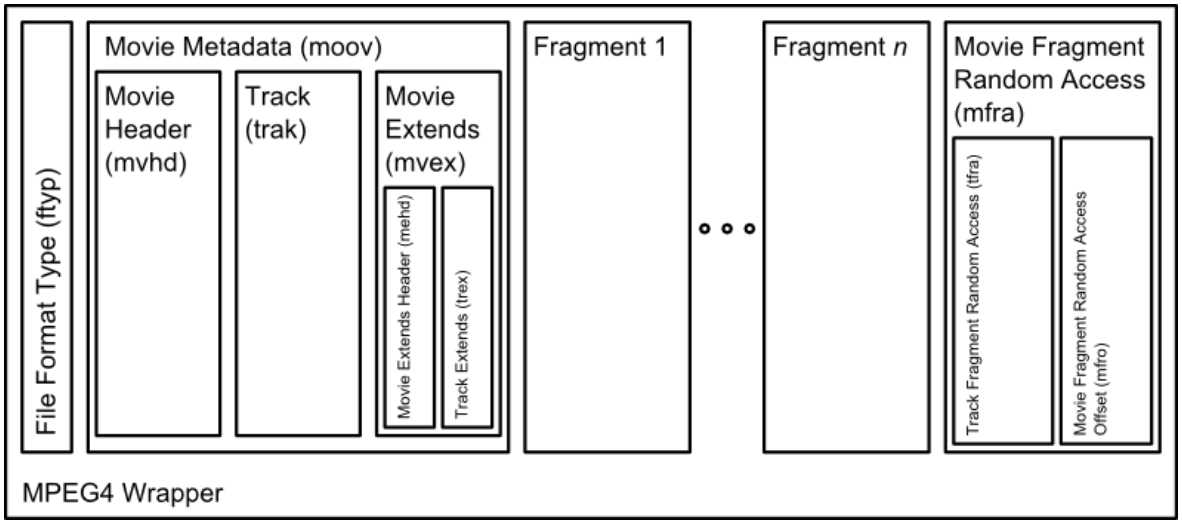


**Figure 7**. Fragmented MP4 File Format

Client programs using Smooth Streaming are generally custom-built interfaces built using Silverlight. Initially, for any sort of playback, the client requests an XML manifest containing information about available aspects of the desired video stream (see Fig. 9): the codecs used to encode and compress the content, resolutions available (with their respective bitrates), and a list of the video chunks with either a start time or duration. [17] The client can then request any fragment from the server using a RESTful URL (see Fig. 8).

```
http://video.foo.com/TITLE.ism/QualityLevels(400000)/Fragments(video=610275114)
```

**Figure 8**. Sample RESTful Address for a Smooth Streaming Video Fragment [5]

```
<SmoothStreamingMedia
  MajorVersion="2"
  MinorVersion="1"
  Duration="5964800000">
  <StreamIndex
    Type="video"
    Name="video"
    Chunks="312"
    QualityLevels="2"
    Url="QualityLevels({bitrate})/Fragments(video={start time})">
    <QualityLevel
      Index="0"
      Bitrate="6000000"
      FourCC="WVC1"
      MaxWidth="1920"
      MaxHeight="1080"
      CodecPrivateData="250000010FDBBE3BF21B8A3BF8EFF18044800000010E5A0040" />
    <QualityLevel
      Index="1"
      Bitrate="4176000"
      FourCC="WVC1"
      MaxWidth="1920"
      MaxHeight="1080"
      CodecPrivateData="250000010FDBBE3BF21B8A3BF8EFF18044800000010E5A0040" />
    <QualityLevel
      Index="0"
      Bitrate="330000"
      FourCC="WVC1"
      MaxWidth="480"
      MaxHeight="272"
      CodecPrivateData="250000010FDB8A3BF21B8A3BF8EFF18044800000010E5A0040" />
    <QualityLevel
      Index="1"
      Bitrate="230000"
      FourCC="WVC1"
      MaxWidth="320"
      MaxHeight="180"
      CodecPrivateData="250000010FDB863BF21B8A3BF8EFF18044800000010E5A0040" />
    <c d="19999968"/>
 </StreamIndex>
  <StreamIndex
    Type="audio"
    Index="0"
    Name="audio"
    Chunks="299"
    QualityLevels="1"
    Url="QualityLevels({bitrate})/Fragments(audio={start time})">
    <QualityLevel
      FourCC="WMAP"
      Bitrate="128000"
      SamplingRate="44100"
      Channels="2"
      BitsPerSample="16"
      PacketSize="5945"
      AudioTag="354"
      CodecPrivateData="1000030000000000000000000000E0000000" />
    <c d="20433560"/>
  </StreamIndex>
</SmoothStreamingMedia>
```

**Figure 9**. Sample Smooth Streaming Client Manifest [17]

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="urn:mpeg:DASH:schema:MPD:2011"
     xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011"
     profiles="urn:mpeg:dash:profile:full:2011"
     type="static"
     mediaPresentationDuration="PT0H9M56.46S"
     minBufferTime="PT15.0S">
    <BaseURL>http://www-itec.uni-klu.ac.at/dash/js/content/</BaseURL>
    <Period start="PT0S">
        <AdaptationSet bitstreamSwitching="true">
            <Representation id="0" codecs="vp8" mimeType="video/webm" width="854"
height="480" startWithSAP="1" bandwidth="329040">
                <SegmentBase>
                    <Initialization sourceURL="sintel_200k.webm" range="0-4767"/>
                </SegmentBase>
                <SegmentList duration="2">
                    <SegmentURL media="sintel_200k.webm" mediaRange="4768-82046"/>
                    <SegmentURL media="sintel_200k.webm" mediaRange="82047-162639"/>
                    <SegmentURL media="sintel_200k.webm" mediaRange="162640-240275"/>
                </SegmentList>
            </Representation>
            <Representation id="0" codecs="vp8" mimeType="video/webm" width="854"
height="480" startWithSAP="1" bandwidth="573804">
                <SegmentBase>
                    <Initialization sourceURL="sintel_500k.webm" range="0-4767"/>
                </SegmentBase>
                <SegmentList duration="2">
                    <SegmentURL media="sintel_500k.webm" mediaRange="4768-122324"/>
                    <SegmentURL media="sintel_500k.webm" mediaRange="122325-280293"/>
                    <SegmentURL media="sintel_500k.webm" mediaRange="280294-435129"/>
                </SegmentList>
            </Representation>
            <Representation id="0" codecs="vp8" mimeType="video/webm" width="854"
height="480" startWithSAP="1" bandwidth="812312">
                <SegmentBase>
                    <Initialization sourceURL="sintel_800k.webm" range="0-4767"/>
                </SegmentBase>
                <SegmentList duration="2">
                    <SegmentURL media="sintel_800k.webm" mediaRange="4768-195920"/>
                    <SegmentURL media="sintel_800k.webm" mediaRange="195921-458392"/>
                    <SegmentURL media="sintel_800k.webm" mediaRange="458393-692430"/>
                </SegmentList>
            </Representation>
            <Representation id="0" codecs="vp8" mimeType="video/webm" width="854"
height="480" startWithSAP="1" bandwidth="1281002">
                <SegmentBase>
                    <Initialization sourceURL="sintel_1400k.webm" range="0-4767"/>
                </SegmentBase>
                <SegmentList duration="2">
                    <SegmentURL media="sintel_1400k.webm" mediaRange="4768-203054"/>
                    <SegmentURL media="sintel_1400k.webm" mediaRange="203055-586870"/>
                    <SegmentURL media="sintel_1400k.webm" mediaRange="586871-949866"/>
                </SegmentList>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

**Figure 10**. Sample MPEG-DASH MPD Manifest (ITEC) [23]

```
<?xml version="1.0"?>
<MPD
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:DASH:schema:MPD:2011"
  xmlns:drm="http://example.net/052011/drm"
  xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 DASH-MPD.xsd"
  type="static"
  mediaPresentationDuration="PT3256S"
  minBufferTime="PT10.00S"
  profiles="urn:mpeg:dash:profile:isoff-on-demand:2011">
  <BaseURL>http://cdn.example.com/movie23453235/</BaseURL>
  <Period>
    <!-- Audio protected with a specified license -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.0x40" lang="en"
      subsegmentStartsWithSAP="1" subsegmentAlignment="true">
      <ContentProtection schemeIdUri="http://example.net/052011/drm">
        <drm:License>http://MoviesSP.example.com/protect?license=kljklsm</drm:License>
        <drm:Content>http://MoviesSP.example.com/protect?content=oyfYvpo</drm:Content>
      </ContentProtection>
      <Representation id="1" bandwidth="64000">
        <BaseURL>audio/en/64.mp4</BaseURL>
      </Representation>
    </AdaptationSet>
    <!-- Audio protected with embedded information defined by 'ZZZZ' -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.0x40" lang="fr"
      subsegmentStartsWithSAP="1" subsegmentAlignment="true">
      <ContentProtection schemeIdUri=" urn:mpeg:dash:mp4protection:2011" value="ZZZZ"/>
      <Representation id="3" bandwidth="64000">
        <BaseURL>audio/fr/64.mp4</BaseURL>
      </Representation>
    </AdaptationSet>
    <!-- Timed text in the clear -->
    <AdaptationSet mimeType="application/ttml+xml" lang="de">
      <Representation id="5" bandwidth="256">
        <BaseURL>subtitles/de.xml</BaseURL>
      </Representation>
    </AdaptationSet>
    <!-- Video protected with a specified license -->
    <AdaptationSet mimeType="video/mp4" codecs="avc1"
      subsegmentAlignment="true" subsegmentStartsWithSAP="2">
      <ContentProtection schemeIdUri="http://example.net/052011/drm">
        <drm:License>http://MoviesSP.example.com/protect?license=jfjhwls</drm:License>
        <drm:Content>http://MoviesSP.example.com/protect?content=mslkfjs</drm:Content>
      </ContentProtection>
      <BaseURL>video/</BaseURL>
      <Representation id="6" bandwidth="256000" width="320" height="240">
        <BaseURL>video256.mp4</BaseURL>
      </Representation>
      <Representation id="7" bandwidth="512000" width="320" height="240">
        <BaseURL>video512.mp4</BaseURL>
      </Representation>
      <Representation id="8" bandwidth="1024000" width="640" height="480">
        <BaseURL>video1024.mp4</BaseURL>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>
```

**Figure 11**. Sample MPD with Encryption Defined (ISO) [23]

Encryption is available through Microsoft's proprietary PlayReady DRM scheme. When initially accessing encrypted content, the Silverlight client contacts the content host's PlayReady license server to request a license for playback. However, PlayReady requires that the content have a secure path between the video player and the output hardware for HD content. Because of this, Silverlight is not supported on open-source clients such as Linux, despite only being a browser plug-in. [18]

### *MPEG-DASH*

Dynamic Adaptive Streaming over HTTP is a technology recently developed by the Moving Picture Experts Group, and was published as an international standard in November 2011 (ISO/IEC 23009-1:2012). [19] DASH is designed to be codec-agnostic, but the specification recommends use of MPEG-4 files or MPEG-2 streams. [20] However, as the specification does not require a particular codec when hosting a stream, there is no guaranteed compatibility with any given MPEG-DASH client, as the client would have to handle virtually any codec in order to maintain constant compatibility. There also remains the question of whether or not the DASH standard will be royalty-free. Some potential client developers, particularly those in the open-source software community, will not build DASH-compatible clients unless it can be implemented using only free/*libre* software. [21] These shortcomings have slowed the adoption of DASH relative to other stream formats.

DASH remains similar to earlier adaptive streaming technologies in that the client requests an XML manifest called a Media Presentation Description (MPD) with information about the properties of the stream (see Fig. 10), and still must request each new segment individually. [22] The MPD can also list multiple different representations of the same media for separate platforms. Though each platform implementing DASH may have different codec requirements, the DASH specification does dictate a common scheme for how encryption must be implemented, so that client platforms with differing built-in DRM processes can decrypt and decode the same stream.

Stream encryption, when present, is defined in the MPD, along with the supported DRM schemes. Different components of the stream can use different encryption and licensing schemes as long as licensing information is provided (see Fig. 11). [19] An MPD can also reference other dynamically created MPDs. This allows for seamless stream splicing. In this manner commercial content that would normally have interstitial advertisements can have the ads easily injected into the video stream. [12] [19]

## Criticisms

The primary drawback to adaptive streaming is precisely what makes it so versatile for users once implemented: encoding several bit-rates. Having several encodings of the same video increases cost, as the copies require more storage and new encoding equipment must be purchased to encode those copies. The encoding process also adds substantial latency for broadcasting live events as the video must be encoded in multiple bitrates and segmented as it is being recorded before those segments can be sent to users as requested. Usually video quality and size are sacrificed to compensate for finite computing power in order to keep the feed as close to live as possible. [9] [24]

Adaptive streaming also does not yet open standard for digital rights management (DRM) for commercial content. Presently, any encryption of content is handled by proprietary implementations. This often results in a CDN having to provide delivery using several control protocols, increasing the complexity and cost of implementation. In the case of Netflix, a different protocol is required depending on the client platform: Microsoft Silverlight for Mac OSX and Windows, and a custom SDK for iOS and Android-based mobile devices. [24]

## Conclusion

For now, the benefits of implementing adaptive streaming for commercial content outweigh the complexity and cost of using the technique. The end result of having such a system is improved quality of service for the end user. Existing broadcast television programming is provided in a manner that is seamless to use and without any playback interruptions as long as the physical broadcasting infrastructure remains intact. Internet-based content streaming is still susceptible to deteriorating bandwidth resulting from higher traffic on a shared connection. For the foreseeable future, as long as bandwidth is a limiting factor, streaming will need to be able to adapt to local conditions in order to remain competitive to television service.

## Abbreviations

AAC – Advanced Audio Coding
AES – Advanced Encryption Standard
CDN – Content Delivery Network
DRM – Digital Rights Management
HTTP – Hypertext Transfer Protocol
HTTPS – Secure Hypertext Transfer Protocol
MPEG – Moving Picture Experts Group
RTMP – Real-Time Messaging Protocol
RTSP – Real-Time Streaming Protocol
SDK – Software Development Kit
UDP – User Datagram Protocol
URI – Uniform Resource Identifier
URL – Uniform Resource Locator

## References

[1] Cisco Systems, Inc., "Cisco Visual Networking Index: Forecast and Methodology, 2012-2017," White Paper, 2013.

[2] Thomas Stockhammer, "Dynamic Adaptive Streaming over HTTP – Standards and Design Principles." In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, New York, NY, 2011, pp. 133-144.

[3] Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner, "An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC." In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, New York, NY, 2010, pp. 157-168.

[4] H. Schulzrinne, A. Rao, and R. Lanphier. (1998, April) Real Time Streaming Protocol (RTSP).

[5] Alex Zambelli, "IIS Smooth Streaming Technical Overview," Microsoft Corporation, 2009.

[6] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano, "Feedback Control for Adaptive Live Video Streaming." In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, New York, NY, 2011, pp. 145-156.

[7] Stefan Lederer, Christopher Müller, and Christian Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments." In *Proceedings of the 4th Workshop on Mobile Video*, New York, NY, 2012, pp. 37-42.

[8] Stefan Lederer, Christopher Müller, and Christian Timmerer, "Dynamic adaptive streaming over HTTP dataset." In *Proceedings of the 3rd Multimedia Systems Conference*, New York, NY, 2012, pp. 89-94.
[9] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP." In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, New York, NY, 2011, pp. 157-168.
[10] David Hassoun. (2010, August) Dynamic streaming in Flash Media Server 3.5. 2013.
[11] R. van Brandenburg, O. van Deventer, F. Le Faucheur, and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)," Internet Engineering Task Force, Informational RFC 6983, July 2013.
[12] Stefan Kaiser, Stefan Pham, and Stefan Arbanowski, "MPEG-DASH enabling adaptive streaming with personalized commercial breaks and second screen scenarios." In *Proceedings of the 11th European Conference on Interactive TV and Video*, New York, NY, 2013, pp. 63-66.
[13] Adobe Systems, Inc., "Adobe's Real Time Messaging Protocol," 2012.
[14] (2013, June) OSMF Wiki. [Online]. http://sourceforge.net/apps/mediawiki/osmf.adobe/index.php?title=Flash_Media_Manifest_%28F4M%29_File_Format
[15] John Crosby. (2011, March) The Kuroko. [Online]. http://www.thekuroko.com/what-is-http-dynamic-streaming/
[16] Ed Pantos and W. May. (2013, April) HTTP Live Streaming. Internet-Draft.
[17] Microsoft Corporation. (2010, March) The Protected Interoperable File Format (PIFF).
[18] Microsoft Corporation, "Microsoft PlayReady Content Access Technology," White Paper, 2008.
[19] International Organization for Standardization. (2012, April) ISO/IEC 23009-1: Information technology — Dynamic adaptive streaming over HTTP.
[20] Christian Timmerer and Carsten Griwodz, "Dynamic adaptive streaming over HTTP: from content creation to consumption." In *Proceedings of the 20th ACM International Conference on Multimedia*, New York, NY, 2012, pp. 1533-1534.
[21] MPEG-DASH Industry Forum. (2013) Overview of DASH-MPEG Standard. [Online]. http://dashif.org/mpeg-dash/
[22] Christopher Müller and Christian Timmerer, "A VLC media player plug-in enabling dynamic adaptive streaming over HTTP." In *Proceedings of the 19th ACM International Conference on Multimedia* , New York, NY, 2011, pp. 723-726.
[23] Scott Sheridan. (2011, July) Realeyes. [Online]. http://www.realeyes.com/blog/2011/07/27/http-dynamic-streaming-part-1-an-introduction/
[24] Q. Wu and R. Huang. (2010, October) Problem Statement for HTTP Streaming. Internet-Draft.

---

* **TED D. MONCHAMP** is a software developer at Creative Logistics Solutions. He graduated from Rivier University in 2011 with a B.S. in Computer Science, and is currently pursuing a Master's in Computer Science there as well. In the little free time he has left over, Ted mentors students interested in programming with Londonderry High School's FIRST Robotics team, and enjoys studying chemistry and foreign languages.