AN OVERVIEW OF HTTP ADAPTIVE STREAMING PROTOCOLS FOR TV EVERYWHRE DELIVERY

Yuval Fisher
RGB Networks

Abstract

In this paper we review the advantages of adaptive HTTP streaming and detail its usefulness in delivering content on both managed and unmanaged networks. We review the differences between these protocols and discuss their strengths and weaknesses. Additionally, we'll give a detailed look at the packaging and delivery mechanisms for TV Everywhere in which we explain the implications for storage, CDN distribution, ad insertion and overall architecture.

INTRODUCTION

The traditional television viewing experience has clearly changed as viewing video online, on a tablet or smartphone, or on the living room TV thanks to Internet-delivered content is increasingly commonplace. With considerable speed, consumers have passed the early-adopter phase of TV Everywhere and today an ever increasing number expect any program to be immediately available on any viewing device and over any network connection. What's more, they expect this content to be of the same high quality they experience with traditional television services. Regardless of whether this explosion of multiscreen IP video is a threat or an opportunity for cable operators and other traditional video service providers (VSPs), it's clear that it's here to stay.

Despite advancements in core and last mile bandwidth in the past few years, the bandwidth requirement of video traffic is prodigious. Combining this with the reality that the Internet as a whole is not a managed quality-of-service (QoS) environment, means new ways to transport video must be employed to ensure that the high quality of experience (QoE) consumers enjoy with their managed TV delivery networks is the same as what they experience across all their devices and network connections.

To address the conundrum of ensuring optimal quality despite the bandwidth-hungry nature of video and the lack of QoS controls in unmanaged networks, Apple, Microsoft, Adobe and MPEG have developed adaptive delivery protocols. These have been broadly adopted by cable operators and other VSPs, including traditional players and new entrants. The result is that networks are now equipped with servers that can 'package' high-quality video content from live streams or file sources for transport to devices that support these new delivery protocols.

This paper reviews the four primary HTTP adaptive streaming technologies: Apple's HTTP Live Streaming (HLS), Microsoft Silverlight Smooth Streaming (MSS), Adobe's HTTP Dynamic Streaming (HDS) and MPEG's Dynamic Adaptive Streaming over HTTP (DASH).

The paper is divided into three key sections. The first is an overview of adaptive HTTP streaming which reviews delivery architectures, describes its strengths and weaknesses, and examines live and video-on-demand (VOD) delivery. The second section looks at each technology, details how they work and notes how each technology differs from the others. The third and final section looks at specific features and describes how they are implemented or deployed. In particular, this last section focuses on:

- Delivery of multiple audio channels
- Encryption and DRM
- Closed captions / subtitling
- Ability to insert ads
- Custom VOD playlists
- Trick modes (fast-forward/rewind, pause)
- Fast channel change
- Failover due to upstream issues
- Stream latency
- Ability to send other data to the client, including manifest compression

ADAPTIVE BITRATE HTTP VIDEO DELIVERY

In Adaptive Bitrate (ABR) HTTP streaming, the source video – whether a live stream or a file – is encoded into discrete file segments known as ‘fragments’ or ‘chunks.’ The contents of these fragment files can include video data, audio data or such other data such as subtitles, program information or other metadata. These data may be multiplexed in the file fragment or can be separated into distinct fragment files. The fragments are hosted on an HTTP server from which they are served to clients. A sequence of fragments is called a ‘profile.’ The same content may be represented by different profiles that may differ in bitrate, resolution, codecs or codec profile/level.

Clients play the stream by requesting fragments from the HTTP server. The client then plays the fragments contiguously as they are downloaded. If the audio, video or other data are stored in separate fragment sequences, all are downloaded to form the video playback.

The video in each fragment file is typically encoded as H.264, though HEVC or other video codecs are possible. AAC is generally used to encode the audio data, but again, other formats are also used. Fragments typically represent 2 to 10 seconds of video. The stream is broken into fragments at video Group of Pictures (GOP) boundaries that begin with an IDR frame, which is a frame that can be independently decoded without dependencies on other frames. In this way, a client can play a fragment from the beginning without any dependence on previous or following fragments.

As the name suggests, adaptive delivery enables a client to ‘adapt’ to varying network conditions by selecting video fragments from profiles that are better suited to the conditions at that moment. Computing the available network bandwidth is easily accomplished by the client, which compares the download time of a fragment with its size. Using a list of available profile bitrates (or resolutions or codecs), the client can determine if the bandwidth available is sufficient for it to download fragments from a higher bitrate/resolution profile or if it needs to change to a lower bitrate/resolution profile. This list of available profiles is called a ‘manifest’ or ‘playlist.’ The client can quickly adapt to fluctuating bandwidth – or other network conditions – every few seconds as it continually performs bandwidth calculations at each fragment download. Local CPU load and the client’s ability to play back a specific codec or resolution are factors – in addition to available bandwidth – that may affect the client’s choice of profile. For example, a manifest file may reference a broad collection of profiles with a wide selection of codecs and resolutions, but the client will know that it can only play back certain profiles and therefore only request fragments from those profiles.

This model enables delivery of both live and on-demand-based sources. A manifest file is provided to the client in both scenarios. The manifest lists the bitrates (and potentially other data) of the available stream profiles so the client can determine how to download the chunks; that is, the manifests tells the client what URL to use to fetch fragments from specific profiles. In the case of an on-demand file request, the manifest contains information on every fragment in the content. This is not possible when it comes to live streaming. DASH, HLS and HDS deliver 'rolling window' manifest data that contains references to the last few available fragments, as shown in Figure 1. Continual updating of the client's manifest is necessary to know about the most recently available chunks. For MSS, a rolling window-type manifest isn't necessary because MSS delivers information in each fragment that lets the client access subsequent fragments.

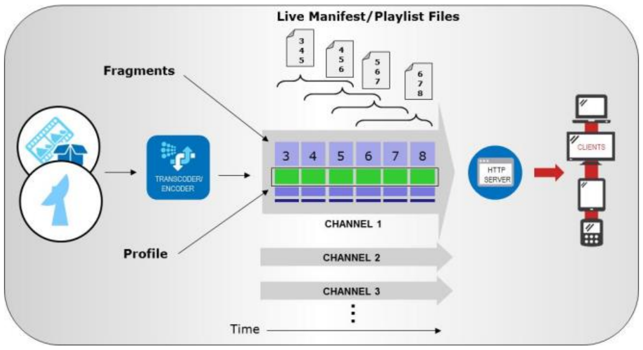


Figure 1. The content delivery chain for a live adaptive HTTP stream. The client downloads the 'rolling window' manifest files which references the latest available chunks. The client uses these references to download chunks for sequential play back. In the figure, the first manifest refers to chunks 3, 4 and 5, which are available in multiple bitrates. The playlist is updated as new chunks become available to reference the latest available chunks.

The advantages of adaptive HTTP streaming include:

- Being able to utilize generic HTTP caches, proxies and content delivery networks, as are used for web traffic;
- Content delivery is dynamically adapted to the weakest link in the end-to-end delivery chain, including highly varying last mile conditions;
- Playback control functions can be initiated using the lowest bitrate fragments and then transitioned to higher bitrates, enabling viewers to enjoy fast start-up and seek times;
- The client can control bitrate switching taking into account CPU load, available bandwidth, resolution, codec and other local conditions;
- Firewalls and NAT do not hinder HTTP delivery.

There are also a few issues with adaptive HTTP streaming:

- The end-to-end latency of live streams is increased as clients must buffer a few fragments to ensure that their input buffers aren't starved;
- HTTP is based on TCP and TCP recovers well when packet loss is low, meaning that video playback has no artifacts caused by missing data. However, TCP can completely fail when packet loss rises. Because of this clients usually experience good quality playback or the playback stops entirely – there is no middle ground. This is as opposed to quality that degrades proportionally to the amount of packet loss in the delivery network. Typically the Internet is sufficiently reliable that

the benefit of completely clean video at low packet drop rates outweighs the value of some poor quality video at high packet drop rates (e.g. with UDP-based streaming).

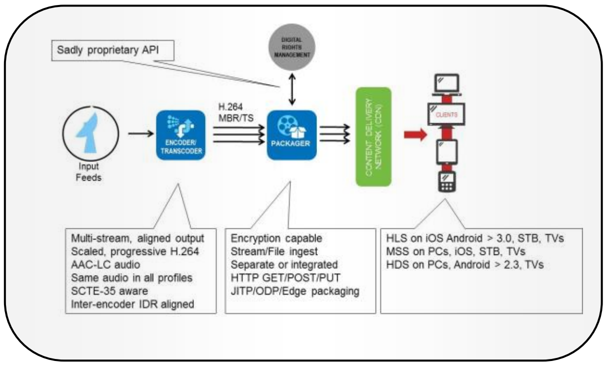

Sadly proprietary API
DIGITAL RIGHTS MANAGEMENT
Input Feeds
ENCODER/ TRANSCODER
H.264 MBR/TS
PACKAGER
CONTENT DELIVERY NETWORK (CDN)
CLIENTS
Multi-stream, aligned output
Scaled, progressive H.264
AAC-LC audio
Same audio in all profiles
SCTE-35 aware
Inter-encoder IDR aligned
Encryption capable
Stream/File ingest
Separate or integrated
HTTP GET/POST/PUT
JITP/ODP/Edge packaging
HLS on iOS Android > 3.0, STB, TVs
MSS on PCs, iOS, STB, TVs
HDS on PCs, Android > 2.3, TVs

Figure 2. The components of an HTTP streaming system.

Components of Video Delivery

The key components in an adaptive HTTP streaming data flow consist of an encoder or transcoder, a packager (sometimes called a ‘segmenter’ or a ‘fragmenter’) and a CDN. In this section we will discuss the features of these components (see Figure 2) as related to adaptive streaming.

The Encoder/Transcoder

The ingestion and preparation of content for segmentation is the responsibility of the transcoder – or encoder, if the input is not already encoded in another format. The following features require support by the transcoder:

- The transcoder must de-interlace the input as the output video must be in progressive format.
- The video must be encoded into H.264 (or HEVC).
- The output video must be scaled to resolutions suitable for the client devices.
- The different output profiles must be IDR-aligned so that client playback of the chunks created from each profile is continuous and smooth.
- Audio must be transcoded into the AAC codec most often used by DASH, HLS, HDS and MSS.
- The same encoded audio stream generally must be streamed on all the output video profiles; to avoid clicking artifacts during client-side profile changes.
- If SCTE-35 is used for ad insertion, it is recommended that the transcoder add IDR frames at the ad insertion points to prepare the video for ad insertion. Fragment boundaries can then be aligned with the ad insertion points so that ad insertion is accomplished by the substitution of fragments, as compared to traditional stream splicing.
- An excellent fault tolerance mechanism allows two transcoders ingesting the same input to create identically IDR-aligned output. This allows the creation of a redundant backup of encoded content so that the secondary transcoder can seamlessly backup the primary transcoder should it fail for any reason.

Because consumers’ QoE requires multiple different profiles for the client to choose from, it is best that the encoder be able to output a large number of different profiles for each input. Deployments may use anywhere from 4 to 16 different output profiles for each input. Naturally, the more profiles there are means the operator can support more devices and deliver a better user experience. The table below shows a typical use case for the different output profiles:

Width	Height	Video Bitrate
1920	1080	5 Mbps
1280	720	3 Mbps
960	540	1.5 Mbps
864	486	1.25 Mbps
640	360	1.0 Mbps
640	360	750 kbps
416	240	500 kbps
320	180	350 kbps
320	180	150 kbps

The Packager

As its name implies, the packager takes the transcoder's output and packages the video for a specific delivery protocol. Ideally, a packager will have the following features and capabilities:

- Encryption – the packager should be able to encrypt the outgoing chunks in a format compatible with the delivery protocol.
- Integration with third party key management systems – the packager should be able to receive encryption keys from a third party key management server that is also used to manage and distribute the keys to the clients.
- Live and on-demand ingest – the packager should be capable of ingesting both live streams and files, depending on whether the workflow is live or on-demand.
- Multiple delivery methods – the packager should support multiple ways to deliver the chunks, either via HTTP pull or by pushing the chunks using a network share or HTTP PUT/POST.

The CDN

CDNs do not need to be specialized for HTTP streaming nor are any special streaming servers required. For live delivery, it is ideal to set the CDN to rapidly age out older chunks as there is no need to keep them around long. A minute is usually enough, but the actual duration is dependent upon on the duration of the chunks and latency in the client.

Note that the number of chunks can be very large. For example, a day's worth of 2-second chunks delivered in 10 different profiles for 100 different channels creates 43 million files! Clearly the CDN must also be capable of handling a large number of files.

The Client

Obviously HLS is available on all iOS devices, but its availability on Windows devices is only thanks to third party products that are not always complete or sufficiently robust. Android include HLS natively, though not all features are well supported, for example stream discontinuity indications. MSS on a PC requires installation of a Silverlight runtime client. However, there are native Smooth Streaming clients for multiple devices, including iOS devices. HDS is native to Flash 10.1 and later releases and is also supported on some smart TVs and STBs. Adobe and Microsoft have announced that they will support DASH.

Despite being supported by a number of clients on various platforms, DASH is not yet experiencing the widespread adoption that the other protocols are. Though, it should be noted that some deployments of HbbTV utilize DASH.

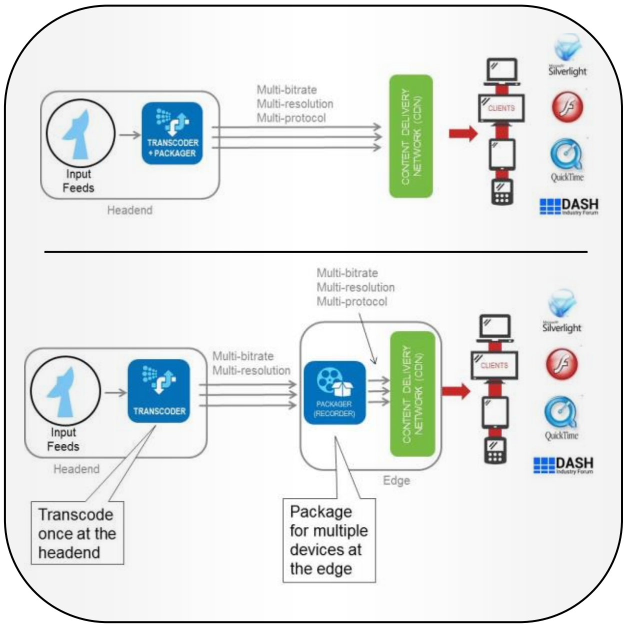


Figure 3. Integrated and remote segmentation of streams: When multiple formats are used, segmenting closer to the edge of the network (top) saves core bandwidth, as streams can be delivered once and packaged at the edge into multiple delivery formats. However, if the core network is susceptible to packet loss, segmenting at the core ensures that segments will always be delivered to the CDN (bottom).

Workflow Architecture

A flexible architecture in which the transcoder and packager can be separate is ideal. The key benefit of this approach is that the input video only needs to be transcoded once at the core, then delivered to the network edge where it's packaged into multiple formats. Without this separation, all the final delivery formats must be delivered over the core network, unnecessarily increasing its bandwidth utilization. This is shown in Figure 3.

Additional Adaptive HTTP Streaming Technologies

In addition to the "Big 4" there are several other adaptive streaming technologies, most notably:

- EchoStar acquired Move Networks and has integrated Move's technology into their home devices. Move played a major role in popularizing adaptive HTTP streaming and has multiple patents on the technology (though chunked streaming was used before Move popularized it).
- 3GPP's Adaptation HTTP Streaming (AHS) is part of 3GPPs rel 9 specification (see [AHS]). And 3GPP rel 10 is working on a specification called DASH as well.
- The Open TV Forum has developed its own HTTP Adaptive Streaming (HAS) specification (see [HAS]).
- MPEG Dynamic Adaptive Streaming over HTTP (DASH) is based on 3GPP's AHS and the Open TV Forum's HAS and is completed. It specifies use of either fMP4 or transport stream (TS) chunks and an XML manifest (called the media presentation description or MPD) that can behave similarly to both MSS or HLS. As a kind of combination of MSS and HLS, DASH allows for multiple scenarios, including separate or joined streaming of audio, video and data, as well as encryption. However, it makes clients complex to implement due to its generality, which is as much a drawback as an advantage. To address this issue, the DASH Industry Forum (see [DASHIF]) created the DASH-264 (and DASH-265)

specifications which make specific profile suggestions to create a more readily implementable and interoperable specification, focused on the Base Media File Format (BMFF) using H.264 (or H.265) and AAC audio. Questions about Intellectual Property rights may be slowing its adoption, but DASH-264/265 has the potential to become the format of choice in the future.

- Though some DRM vendors still have their own variation of these schemes, none appear to have any significant traction.

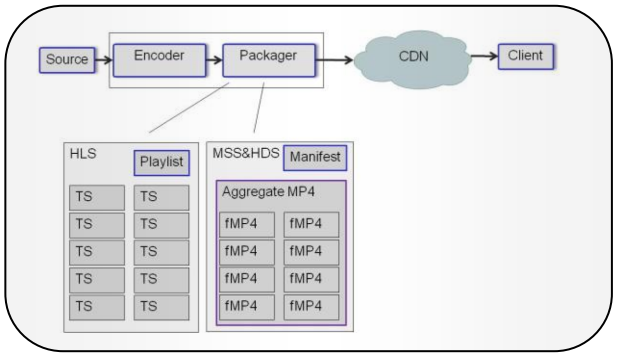


Figure 4. A comparison of HLS and MSS/HDS: The latter can create aggregate formats that can be distributed on a CDN for VOD, whereas for live video, all distribute chunks on the CDN. MSS allows audio and video to be aggregated and delivered separately, but HDS and HLS deliver these together.

THE INTERNALS OF ADAPTIVE STREAMING

Let's now review some of the details of HLS, HDS, MSS and DASH. Each protocol has its own unique strengths and weaknesses, which will be reviewed in the following sections.

Apple HTTP Live Streaming (HLS)

Interestingly, Apple chose not to use the ISO MPEG file format (which is based on its own MOV file format) in its adaptive streaming technology, unlike Adobe and Microsoft. Instead, HLS takes an MPEG-2 transport stream and segments it to a sequence of MPEG-2 TS files which encapsulate the audio and video. These segments are placed on any HTTP server along with the playlist files. The playlist (or index) manifest file is a text file (based on Winamp's original m3u file format) with an m3u8 extension. Full details can be found in [HLS].

HLS defines two types of playlist files: normal and variant. The normal playlist file lists URLs that point to chunks that should be played sequentially. The variant playlist files points to a collection of different normal playlist files, one for each output profile.

Metadata is carried in the playlist files as comments – lines preceded by '#'. In the case of normal playlist files, this metadata includes a sequence number that associate chunks from different profiles, chunk duration information, a directive signaling whether chunks can be cached, the location of decryption keys, the type of stream and time information. In the case of a variant playlist the metadata includes the bitrate of the profile, its resolution, its codec and an ID that can associate different encodings of the same content.

Figure 5 and Figure 6 show a sample HLS variant playlist file and normal playlist file. For an HLS client to know the URLs of the most recently available chunks, it's necessary for a playlist file corresponding to a live stream to be repeatedly downloaded. The playlist is downloaded every time a chunk is played, and thus, in order to minimize the number of these requests, Apple recommends a duration of 10 seconds, which is relatively long. However, the size of the playlist file is small compared with any video content, and the client maintains an open TCP connection to the server, so that this network load is not significant. Shorter chunk durations can thus be used, so the client can more quickly adapt to bitrates. VOD playlists are distinguished from live playlists by the `#EXT-X-PLAYLIST-TYPE` and `#EXT-X-ENDLIST` tags.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=531475
mystic_S1/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=381481
mystic_S2/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=531461
mystic_S3/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=781452
mystic_S4/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1031452
mystic_S5/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1281452
mystic_S6/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1531464
mystic_S7/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=3031464
mystic_S8/mnf.m3u8
```

Figure 5. An HLS variant playlist file showing eight output profiles with different bitrates. The URLs for the m3u8 files are relative, but could include the leading 'http://…'. In this example, each profile's playlist is in a separate path component of the URL**.**

Only the HLS protocol does not require chunks to start with IDR frames. It can download chunks from two profiles and switch the decoder between profiles on an IDR frame that occurs in the middle of a chunk. The downside to this is the requirement for extra bandwidth as two chunks corresponding to the same portion of video are downloaded simultaneously.

HLS Considerations

The advantages of HLS include:

- It is a simple protocol and is easily modified. The playlists can be easily accessed and their text format makes modification for applications such a re-broadcast or ad insertion simple.
- The use of TS files means that there is a rich ecosystem for testing and verifying file conformance.
- TS files can carry SCTE 35 cues, ID3 tags (see [HLSID3]) or other such metadata.
- Monetizing HLS is more easily accomplished as it's native to popular iOS devices, the users of which are accustomed to paying for apps and other services.

```
#EXTM3U
#EXT-X-KEY:METHOD=NONE
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:494

#EXT-X-KEY:METHOD=NONE
#EXTINF:10,505.ts
505.ts
#EXTINF:10,506.ts
506.ts
#EXTINF:10,507.ts
507.ts
```

Figure 6. An HLS playlist file from a live stream showing the three latest available TS chunks. The #EXT-X-MEDIA-SEQUENCE:494 is used by the client to keep track of where it is in the linear playback. The fragment name carries no streaming-specific information. The #EXT-X-TARGETDURATION:10 tag is the expected duration (10 seconds) of the chunks, though durations can vary. The #EXT-X-KEY:METHOD=NONE tag shows that no encryption was used in this sequence. The #EXTINF:10 tags show the duration of each segment. As in the variant playlist file, the URLs are relative to the base URL used to fetch the playlist.

The disadvantages of HLS include:

- HLS is not supported natively on Windows OS platforms.
- Apple's aggregate format stores all the fragments in one TS file and uses byte-range URL requests to pull out the fragment data. Unfortunately CDNs are sometimes unable to cache based on such requests, which limits the usefulness of this aggregation format. Without an aggregation format, HLS must store each fragment as a separate file, so that many files must be created. For example, a day's worth of programming for a single channel requires almost 70,000 files,

assuming eight profiles with 10-second chunk duration. Clearly the managing of such a large collection of files is not convenient.

- HLS is an ecosystem in which different iOS clients have different capabilities, and this limits the value of the later improvements to HLS. This because HLS has evolved from requiring fragments that mux audio and video to allowing separate fragments (as well as other developing features).

Microsoft's Silverlight Smooth Streaming (MSS)

Silverlight Smooth Streaming delivers streams as a sequence of ISO MPEG-4 files (see [MSS] and [MP4]). Usually these are pushed by an encoder to a Microsoft IIS server (using HTTP POST), which aggregates them for each profile into an 'ismv' file for video and an 'isma' file for audio. The IIS server also creates an XML manifest file that contains information about the bitrates and resolutions of the available profiles (see Figure 7). When the request for the manifest comes from a Microsoft IIS server, it has a specific format:

```
http://{serverName}/{PublishingPointPath}/{Pub
        lishingPointName}.isml/manifest
```

The `PublishingPointPath` and `PublishingPointName` are derived from the IIS configuration.

In MSS, the manifest files contain information that allows the client to create a RESTful URL request based on timing information in the stream, which differs from HLS in which URLs are given explicitly in the playlist. For live streaming, the client computes the URLs for the chunks in each profile directly, rather than repeatedly downloading a manifest. The segments are extracted from the ismv and isma files and served as 'fragmented' ISO MPEG-4 (fMP4) files. MSS (optionally) separates the audio and video into separate chunks and combines them in the player.

```
<SmoothStreamingMedia MajorVersion="2"
MinorVersion="0" TimeScale="10000000" Duration="0"
LookAheadFragmentCount="2" IsLive="TRUE"
DVRWindowLength="300000000">
    <StreamIndex Type="video" QualityLevels="3"
        TimeScale="10000000" Name="video" Chunks="14"
        Url="QualityLevels({bitrate})/Fragments(vide
        o={start time})" MaxWidth="1280"
        MaxHeight="720" DisplayWidth="1280"
        DisplayHeight="720">
        <QualityLevel Index="0" Bitrate="350000"
        CodecPrivateData="00000001274D401F9A6282833F
        3E022000007D20001D4C12800000000128EE3880"
        MaxWidth="320" MaxHeight="180" FourCC="H264"
        NALUnitLengthField="4"/>
        <QualityLevel Index="1" Bitrate="500000"
        CodecPrivateData="00000001274D401F9A628343F6
        022000007D20001D4C12800000000128EE3880"
        MaxWidth="416" MaxHeight="240" FourCC="H264"
        NALUnitLengthField="4"/>
        <QualityLevel Index="2" Bitrate="750000"
        CodecPrivateData="00000001274D401F9A6281405F
        F2E022000007D20001D4C1280000000128EE3880"
        MaxWidth="640" MaxHeight="360" FourCC="H264"
        NALUnitLengthField="4"/>
        <c t="2489409751000"/>
        <c t="2489431105667"/>
        <c t="2489452460333"/>
        <c t="2489473815000"/>
        <c t="2489495169667"/>
        <c t="2489516524333"/>
        <c t="2489537879000"/>
        <c t="2489559233667"/>
        <c t="2489580588333" d="21354667"/>
    </StreamIndex>
    <StreamIndex Type="audio" QualityLevels="2"
        TimeScale="10000000" Language="eng"
        Name="audio_eng" Chunks="14"
        Url="QualityLevels({bitrate})/Fragments(audi
        o_eng={start time})">
        <QualityLevel Index="0" Bitrate="31466"
        CodecPrivateData="1190" SamplingRate="48000"
        Channels="2" BitsPerSample="16"
        PacketSize="4" AudioTag="255" FourCC="AACL"/>
        <QualityLevel Index="1" Bitrate="31469"
        CodecPrivateData="1190" SamplingRate="48000"
        Channels="2" BitsPerSample="16"
        PacketSize="4" AudioTag="255" FourCC="AACL"/>
        <c t="2489408295778"/>
        <c t="2489429629111"/>
        <c t="2489450962444"/>
        <c t="2489472295778"/>
        <c t="2489493629111"/>
        <c t="2489514962444"/>
        <c t="2489536295778"/>
        <c t="2489557629111"/>
        <c t="2489578962444" d="21333334"/>
        </StreamIndex>
</SmoothStreamingMedia>
```

Figure 7. A sample MSS manifest file. The elements with 't="248…"' specify the time stamps of chunks that the server has and is ready to deliver. These are converted to Fragment timestamps in the URL requesting an fMP4 chunk. The returned chunk holds time stamps of the next chunk or two (in its UUID box), so that the client can continue fetching chunks without requesting a new manifest.

The URLs below show typical requests

for video and audio. The `QualityLevel` indicates the profile and the `video=` and `audio-eng=` indicate the specific chunk requested. The `Fragments` portion of the request is given using a time stamp (in hundred nanosecond units) that the IIS server uses to extract the correct chunk from the aggregate MP4 audio and/or video files.

```
      http://sourcehost/local/2/mysticSmooth.isml/Qu
alityLevels(350000)/Fragments(video=2489452460333)
      http://sourcehost/local/2/mysticSmooth.isml/Qu
alityLevels(31466)/Fragments(audio-eng=2489450962444)
```

In the VOD case, the manifest files contain timing and sequence information for all the chunks in the content. The player uses this information to create the URL requests for the audio and video chunks.

It is important to recognize that the use of IIS as the source of the manifest and fMP4 files doesn't prohibit using standard HTTP servers in the CDN. The CDN can still cache and deliver the manifest and chunks as it would any other files. More information about MSS can be found at Microsoft (see [SSTO]) and various excellent blogs of the developers of the technology (see [SSBLOG]).

MSS Considerations

The advantages of MSS include:

- IIS creates an aggregate format for the stream, so that a small number of files can hold all the information for the complete smooth stream.
- IIS has useful analysis and logging tools, as well as the ability to deliver more MSS and HLS content directly from the IIS server.
- Rapid adaptation during HTTP stream playback is achieved when the recommended use of a small chunk size is followed. The delivery of different audio tracks requires only a manifest file change due to video and audio files being segregated.
- The aggregate file format supports multiple data tracks that can be used to store metadata about ad insertion, subtitling, etc.

The disadvantages of MSS include:

- The network is data flow is slightly more complex and an extra point of failure is added due to need to place an IIS server in the flow.
- On PCs, MSS requires installation of a separate Silverlight plug-in.

```
<manifest>
    <id>USP</id>
    <startTime>2006-07-24T07:15:00+01:00</startT
    ime>
    <duration>0</duration>
    <mimeType>video/mp4</mimeType>
    <streamType>live</streamType>
    <deliveryType>streaming</deliveryType>
    <bootstrapInfo profile="named"
    url="mysticHDS.bootstrap"/>
    <media
    url="mysticHDS-audio_eng=31492-video=3000000
    -" bitrate="3031" width="1280" height="720"/>
    <media
    url="mysticHDS-audio_eng=31492-video=1500000
    -" bitrate="1531" width="960" height="540"/>
    <media
    url="mysticHDS-audio_eng=31492-video=1250000
    -" bitrate="1281" width="864" height="486"/>
    <media
    url="mysticHDS-audio_eng=31492-video=1000000
    -" bitrate="1031" width="640" height="360"/>
    <media
    url="mysticHDS-audio_eng=31492-video=750000-
    " bitrate="781" width="640" height="360"/>
    <media
    url="mysticHDS-audio_eng=31492-video=500000-
    " bitrate="531" width="416" height="240"/>
    <media
    url="mysticHDS-audio_eng=31469-video=350000-
    " bitrate="381" width="320" height="180"/>
</manifest>
```

Figure 8. A sample HDS manifest file.

Adobe HTTP Dynamic Streaming (HDS)

Adobe HDS uses elements of both HLS and MSS as it was defined after them (see [HDS]). In HDS, an XML manifest file (of file type f4m) contains information about the available profiles (see Figure 8 and [F4M]). Like HLS, the HDS client repeatedly downloads data that allows it to know the URLs of the available chunks -- in HDS this is called the bootstrap information. This bootstrap information isn’t human-readable as it is in a binary format. As in MSS, segments are encoded as fragmented MP4 files containing audio and video information in a single file. HDS chunk requests have the form:

```
http://server_and_path/QualityModifierSeg’segm
   ent_number’-Frag’fragment_number’
```

where the segment and fragment number together define a specific chunk. As in MSS, an aggregate (f4f) file format is used to store all the chunks and extract them when requested.

HDS Considerations

The advantages of HDS include:

- The Flash client is available on multiple devices and is installed on almost every PC worldwide.
- HDS is a part of Flash and can make use of Flash’s environment and readily available developer base.

The disadvantages of HDS are:

- HDS is relatively new and could suffer more from stability issues than its more mature counterparts.
- Deploying a stable ecosystem is difficult due to Adobe’s Flash rapidly changing access roadmap.
- The ecosystem of partners offering compatible solutions is limited due to the binary format of the bootstrap file.

Dynamic Adaptive HTTP Streaming (DASH)

DASH is the most feature complete and complex of all the protocols, as it incorporates many features similar to those in HLS and MSS. It can make use of both TS fragments and fMP4 fragments, and it supports both repeated manifest downloads or URLs derivable from a template. Like MSS and HDS, the DASH manifest, or MPD, uses an XML format. Figure 9 shows a sample DASH MPD.

```
<?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"
type="static"
mediaPresentationDuration="PT12M34.041388S"
minBufferTime="PT10S"
profiles="urn:mpeg:dash:profile:isoff-live:2011">
<Period>
<AdaptationSet mimeType="audio/mp4"
 segmentAlignment="0" lang="eng">
 <SegmentTemplate timescale="10000000"
    media="audio_eng=$Bandwidth$-$Time$.dash"
    initialisation=" audio_eng=$Bandwidth$.dash">
 <SegmentTimeline>
   <S t="667333" d="39473889" />
   <S t="40141222" d="40170555" />
   ...
   <S t="7527647777" d="12766111" />
  </SegmentTimeline>
 </SegmentTemplate>
<Representation id="audio_eng=96000" bandwidth="96000"
    codecs="mp4a.40.2" audioSamplingRate="44100" />
</AdaptationSet>
  <AdaptationSet mimeType="video/mp4"
    segmentAlignment="true" startWithSAP="1"
    lang="eng">
   <SegmentTemplate timescale="10000000"
     media="video=$Bandwidth$-$Time$.dash"
     initialisation="video=$Bandwidth$.dash">
   <SegmentTimeline>
     <S t="0" d="40040000" r="187" />
     <S t="7527520000" d="11678333" />
   </SegmentTimeline>
  </SegmentTemplate>
<Representation id="video=299000" bandwidth="299000"
    codecs="avc1.42C00D" width="320" height="180" />
<Representation id="video=480000" bandwidth="480000"
    codecs="avc1.4D401F" width="512" height="288" />
<Representation id="video=4300000" bandwidth="4300000"
    codecs="avc1.640028" width="1280" height="720" />
</AdaptationSet>
</Period>
</MPD>
```

Figure 9. A sample DASH MPD.

DASH Considerations

The advantages of DASH include:

- It is based on an open standard.
- The DASH-264 specification has strong industry support which bodes well for solid interoperability.

The disadvantages of DASH include:

- It's all-encompassing, so different DASH implementations will most likely be incompatible unless they focus on exactly the same specification profiles.
- The Intellectual Property (IP) rights associated with DASH are not completely clear. Normally, the holders of essential patents have (for the most part) participated in the MPEG process for other standards – leading to so-called reasonable and non-discriminatory (RAND) licensing terms for the IP associated with the specification. In the case of DASH, the picture is more murky.

COMPARING FEATURES

Let's now compare the usability of DASH, HLS, HDS and MSS usability in several common usage scenarios.

Delivering Multiple Audio Channels

HLS is now equal to MSS in its ability to deliver audio and video separately and easily, which is thanks to the release of iOS5 which added the ability to deliver audio separately. Delivering audio separately is, of course, important in locales where multiple languages are used and only one is to be consumed. HDS is still primarily used with audio and video muxed together in the fragments. DASH can flexibly separate video and audio.

Encryption and DRM

HLS supports encryption of each TS file, meaning that all of the data contained in the TS file is encrypted and cannot be extracted without the decryption keys. All metadata related to the stream (e.g. the location of the decryption keys) must be included in the playlist file. While functioning well, HLS does not specify a mechanism for authenticating clients to receive the decryption keys. This is considered a deployment issue. Several vendors offer HLS-type encryption, generally with their own twist which makes the final deployment incompatible with other implementations.

Microsoft's PlayReady is used by MSS to provide a complete framework for encrypting content, managing keys and delivering them to clients. Because PlayReady only encrypts the payload of the fragment file, the chunk can carry other metadata. Microsoft makes PlayReady code available to multiple vendors that productize it, and so a number of vendors offer PlayReady capability (in a relatively undifferentiated way).

HDS uses Adobe's Access, which has an interesting twist that simplifies interaction between the key management server and the scrambler that does the encryption. Typically, keys must be exchanged between these two components, and this exchange interface is not standardized. Each pair of DRM and scrambler vendors must implement this pair-wise proprietary API. With Adobe Access however, key exchanges are not necessary as the decryption keys are sent along with the content and are encrypted themselves. Access to those keys is granted at run time, but no interaction between the key management system and scrambler is needed.

DASH allows DRM systems to share keys, encryption algorithm and other parameters via its support for the so-called common encryption format. This enables the same content to be managed by different clients and DRM systems which implement key distribution and other rights management. This represents a major achievement in the breaking up the traditional DRM vendor model, which locks-in users by making content playable only by clients associated with a specific vendor's encryption.

Closed Captions / Subtitling

As of iOS 4.3, HLS can decode and display closed captions (using ATSC Digital Television Standard Part 4 – MPEG-2 Video System Characteristics - A/53, Part 4:2007, see [ATCA]) included in the TS chunks. For DVB teletext, packagers must convert the subtitle data into ATSC format or wait for clients to support teletext data. HLS also supports WebVTT, which holds subtitles in a separate file.

MSS supports data tracks that hold Time Text Markup Language (TTML), a way to specify a separate data stream with subtitle, timing and placement information (see [TTML]). For MSS, packagers need to extract subtitle information from their input and convert it into a TTML track. Microsoft's implementation of MSS client currently offers support for W3C TTML, but not for SMPTE TTML (see [SMPTE-TT]), which adds support for bitmapped subtitles, commonly used in Europe.

HDS supports data tracks that hold subtitles as DFXP file data or as TTML. In a manner similar to MSS, HDS clients can selectively download this data.

Though the DASH specification supports various forms of subtitling, client support is inconsistent.

Of the many things the formats have in common, it's worth noting that none support DVB image subtitles particularly well. HLS can support these using ID3 signaling and proprietary Javascript wrapped around the client. The other formats can also "support" this using proprietary clients as well.

Targeted Ad insertion

HLS is the simplest protocol for chunk-substitution-based ad insertion. With HLS, the playlist file can be modified to deliver different ad chunks to different clients (see Figure 10). The `EXT-X-DISCONTINUITY` tag can tell the decoder to reset (e.g. because subsequent chunks may have different PID values), and only the sequence ID must be managed carefully, so that the IDs line up when returning to the main program. HDS also supports the repeated downloading of bootstrap data used to specify chunks, and this can be modified to create references to ad chunks. However, because the bootstrap data format is binary, and the URLs are RESTful with references to chunk indexes, the process is complex.

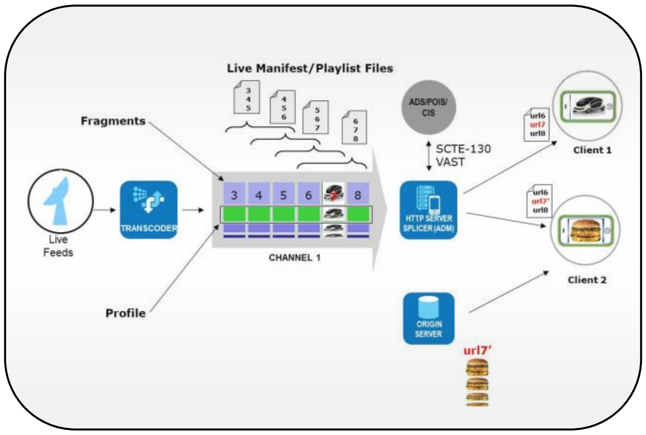


Figure 10. HLS ad insertion in which changes to the playlist file delivered to each client cause each client to make different URL requests for ads and thus receive targeted ad content.

Chunk-based ad insertion for live streams is more complicated with MSS. Because fragments contain timing information used to request the next fragment, all ad fragments must have identical timing to the main content chunks. Regardless, a proxy can redirect RESTful URL fragment requests and serve different fragments to different clients.

MSS and HDS can both deliver control events in separate data tracks. These can trigger client behaviors using the Silverlight and Flash environments, including ad insertion behavior. However, this is beyond the scope of this paper, which is focused on 'in stream' insertion.

A particularly nice feature in DASH allows it to switch from template-based URLs to repeated download of the manifest. The advantage is that the network program can be delivered using templated URLs which stop and switch to downloaded manifests for the targeted ads.

nDVR Mezzanine Format

HLS is clearly the winner when it comes to the stream-to-file capture of adaptive bitrate formats. It captures and recombines fragments much more easily than possible with MSS or HDS. DASH can do the same, in its TS file profile, but not so readily in DASH-264.

Trick Modes (Fast-forward / Rewind)

None of the protocols implement VOD trick modes, such as fast-forward or rewind, well at all. HLS, DASH and HDS offer support for fast-forward and rewind in the protocols, but actual client implementations are non-existent. MSS defines zoetrope images that can be embedded in a separate track. These can be used to show still images from the video sequence and allow viewers to seek a specific location in the video.

Custom VOD Playlists

Being able to take content from multiple different assets and stitch them together to form one asset is particularly convenient. This is readily done in HLS and DASH, where the playlist can contain URLs that reference chunks from different encodings and locations. Unfortunately for MSS and HDS, constructing such playlists is basically impossible because of the RESTful URL name spaces and the references to chunks via time stamp or sequence number.

Fast Channel Change

Adaptive HTTP streaming can download low bitrate fragments initially, making channel 'tune in' times fast. The duration of the fragment directly affects how fast the channel adapts to a higher bandwidth (and higher quality video). This is an advantage for DASH, MSS and HDS, which are tuned to work with smaller fragments, and which tend to work a bit better than HLS.

Failover Due to Upstream Issues

To counter situations in which content is not available, HLS manifests can list failover URLs. The mechanism used in the variant playlist file to specify different profiles can specify failover servers, since the client (starting with iOS 3.1 and later) will attempt to change to the next profile when a profile chunk request returns an HTTP 404 'file not found' code. This is a convenient, distributed redundancy mode.

MSS utilizes a fully programmable run-time client. Similarly, HDS works within the Flash run-time environment. This means that the same failover capabilities can be built into the client. Despite this, neither protocol has a built-in mechanism supporting a generic failover capability.

Each protocol will failover to a different profile if chunks/playlists in a given profile are not available. Potentially, this could enable any of the protocols to be used in a “striping” scenario in which alternate profiles come from different encoders (as long as the encoders output IDR aligned streams), so that an encoder failure causes the client to adapt to a different, available profile.

Stream Latency

Adaptive HTTP clients buffer a number of segments. Typically one segment is currently playing, one is cached and a third is being downloaded. This is done to ensure that the end-to-end latency is minimally about three segment durations long. With HLS recommended to run with 10-second chunks (though this isn’t necessary), this latency can be quite long.

MSS is the sole protocol with a low latency mode in which sub-chunks are delivered to the client as soon as they are available. The client doesn’t need to wait for a whole chunk’s worth of stream to be available at the packager before requesting it, reducing its overall end-to-end latency.

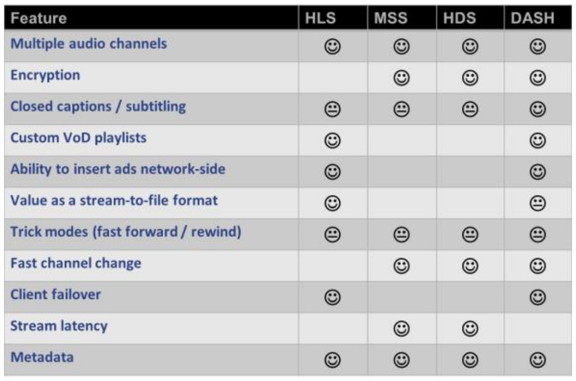
Sending Other Data to the Client (Including Manifest Compression)

HLS, DASH and HDS use playlist and manifest files to send metadata to their clients. MSS and HDS allow data tracks which can trigger client events and contain almost any kind of data. HLS allows a separate ID3 data track to be muxed into the TS chunks. This can be used to trigger client-side events.

MSS and HLS also allow manifest files to be compressed using gzip (as well as internal run-length-type compression constructs) for faster delivery.

CONCLUSION

A snapshot of how these four formats compare is shown in the table below.



Feature	HLS	MSS	HDS	DASH
Multiple audio channels	☺	☺	☺	☺
Encryption		☺	☺	☺
Closed captions / subtitling	😐	😐	😐	☺
Custom VoD playlists	☺			☺
Ability to insert ads network-side	☺			☺
Value as a stream-to-file format	☺			😐
Trick modes (fast forward / rewind)	😐	😐	😐	😐
Fast channel change		☺	☺	☺
Client failover	☺			☺
Stream latency		☺	☺	
Metadata	☺	☺	☺	☺

Though DASH appears strong, in reality its dearth of widely adopted and feature-rich clients makes it look better on paper than reality – notwithstanding the use of DASH by Netflix. Quite understandably, HLS benefits greatly from the ubiquity of iPads and iPhones, and it is the protocol most commonly used by video service providers. Smooth Streaming benefits from the success of the Xbox and the strong brand awareness around Microsoft’s PlayReady DRM, which content owners are comfortable with. Regionally, it has slightly stronger support in Europe, versus the Americas. HDS benefits from the ubiquity of Flash on PCs and laptops, though this does make it the most susceptible to declining usage trends. However, HDS benefits from Adobe Access’s respected DRM, and Adobe’s commitment to focus on DRM and clients across multiple platforms. The conclusion is that at this time, no single format is poised to strongly dominate (and no format shows signs of near-term death). An operator’s final decision must be based on the client devices served, DRM requirements inherited from content owners, and lastly the underlying services delivered to customers. For services that include ad insertion, nDVR and linear TV to iPads, for example, the choice is easy. For other combinations, the decisions are more subtle.

REFERENCES

[HLS] HTTP Live Streaming, R. Pantos, http://tools.ietf.org/html/draft-pantos-http-live-streaming-06

[HLS1] HTTP Live Streaming, http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/HTTPLiveStreaming/_index.html

[MP4] International Organization for Standardization (2003). "MPEG-4 Part 14: MP4 file format; ISO/IEC 14496-14:2003"

[SSBLOG] http://blog.johndeutscher.com/, http://blogs.iis.net/samzhang/ , http://alexzambelli.com/blog/, http://blogs.iis.net/jboch/

[SSTO] Smooth Streaming Technical Overview, http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/

[ATCA] ATSC Digital Television Standard Part 4 – MPEG-2 Video System Characteristics (A/53, Part 4:2007), http://www.atsc.org/cms/standards/a53/a_53-Part-4-2009.pdf

[TTML] Timed Text Markup Language, W3C Recommendation 18 November 2010, http://www.w3.org/TR/ttaf1-dfxp/

[SMPTE-TT] SMPTE Time Text format, https://www.smpte.org/sites/default/files/st2052-1-2010.pdf

[MSS] IIS Smooth Streaming Transport Protocol, http://www.iis.net/community/files/media/smoothspecs/%5BMS-SMTH%5D.pdf

[F4M] Flash Media Manifest File Format Specification, http://osmf.org/dev/osmf/specpdfs/FlashMediaManifestFileFormatSpecification.pdf

[HDS] HTTP Dynamic Streaming on the Adobe Flash Platform, http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf

[AHS] 3GPP TS 26.234: "Transparent end-to-end packet switched streaming service (PSS); Protocols and codecs".

[HAS] OIPF Release 2 Specification - HTTP Adaptive Streaming http://www.openiptvforum.org/docs/Release2/OIPF-T1-R2-Specification-Volume-2a-HTTP-Adaptive-Streaming-V2_0-2010-09-07.pdf

[HLSID3] Timed Metadata for HTTP Live Streaming, http://developer.apple.com/library/ios/#documentation/AudioVideo/Conceptual/HTTP_Live_Streaming_Metadata_Spec/Introduction/Introduction.html

[DVB-BITMAPS] Digital Video Broadcasting (DVB); Subtitling systems, ETSI EN 300 743 V1.3.1 (2006-11), http://www.etsi.org/deliver/etsi_en/300700_300799/300743/01.03.01_60/en_300743v010301p.pdf

[DASHIF] DASH Industry Forum, http://dashif.org/