IIS Smooth Streaming Technical Overview

Author:

***Alex Zambelli**, Media Technology Evangelist*

Microsoft Corporation – March 2009

Contents

Introduction

In October 2008, Microsoft announced that Internet Information Services (IIS) 7.0 would feature a new HTTP-based adaptive streaming extension: **Smooth Streaming**. To promote the new technology, Microsoft also announced an initiative with Akamai and launched a showcase Web site—SmoothHD.com.

Other content delivery networks (CDNs) are expected to announce support for Smooth Streaming in the future.

Smooth Streaming dynamically detects local bandwidth and CPU conditions and seamlessly switches, in near real time, the video quality of a media file that a player receives. Consumers with high-bandwidth connections can experience high definition (HD) quality streaming while others with lower bandwidth speeds receive the appropriate stream for their connectivity, allowing consumers across the board to enjoy a compelling, uninterrupted streaming experience and alleviating the need for media companies to cater to the lowest common denominator quality level within their audience base.

This enables companies to boost brand awareness and advertising revenues by extending average viewing times through higher quality true HD (resolution greater than 720p) experiences. They can also benefit from unprecedented network scalability using distributed HTTP-based Web servers and offer better quality to more customers.

To understand how Smooth Streaming works and what its benefits are, we must first take a look at the history of multiple bit rate media streaming on the Web.

A Brief History of Multiple Bit Rate Streaming

Windows Media Stream Thinning, MBR and Intelligent Streaming

The first effort to adapt streams to client conditions was called *stream thinning*, which Microsoft introduced in 1998 as part of NetShow Services 3.0 and Windows Media® Player 6.1. Stream thinning automatically detected deteriorating network conditions and decreased the video frame rate in response. In dire network conditions, the client could even suspend video playback entirely and stream only audio.

The earliest form of multiple bit rate streaming was introduced by Microsoft in 2000, as part of Windows Media Technologies 4.0 and Windows Media Services 4.1 (in Windows® 2000 Server/Windows NT® Server 4.0) together with Windows Media Player 6.4. The ASF file format allows storage of multiple video and audio tracks inside a single file, and the Windows Media streaming protocols support switching streams during a broadcast. This technology is most commonly referred to as *Multiple Bit Rate ASF*, or simply *MBR*.

In 2002, Microsoft released the Windows Media 9 Series products. Windows Media Services 9 Series (in Windows Server® 2003) and Windows Media Player 9 Series introduced an improved MBR technology dubbed *Intelligent Streaming*. Intelligent Streaming combined bandwidth detection, stream thinning,

MBR ASF, and better image handling in Windows Media Player. Intelligent Streaming, of course, still required the media to be encoded as MBR ASF files with a tool such as Windows Media Encoder 9 Series.

While the technology itself was well designed, its implementations suffered from shortcomings. It was limited to streaming only (no progressive download), and only from Windows Media servers. The encoders did not require that the multiple video streams be temporally aligned, let alone key-frame aligned, which made switching between streams difficult to do in a seamless fashion. Because the media was streamed, and streaming protocols function at constant rates, it was almost impossible to accurately predict overall client bandwidth—particularly in a timely fashion. By the time poor network conditions were detected, it was usually already too late—the Player often went through several iterations of re-buffering before finally downgrading the bit rate.

The Shift to HTTP-Based Delivery

One of the trends that have emerged in the streaming media industry over the past several years has been a steady shift away from classic streaming protocols (RTSP, MMS, RTMP, etc.) back to plain HTTP download. There are many examples of this trend today with community video sharing sites that employ only HTTP progressive download for media delivery.

There are several strong reasons for this industry trend:

- Web download services have traditionally been less expensive than media streaming services offered by CDNs and hosting providers.
- Media protocols often have difficulty getting around firewalls and routers because they are commonly based on UDP sockets over unusual port numbers. HTTP-based media delivery has no such problems because firewalls and routers know to pass HTTP downloads through port 80.
- HTTP media delivery doesn't require special proxies or caches. A media file is just like any other file to a Web cache.
- It's much easier and cheaper to move HTTP data to the edge of the network, closer to users.

Even though streaming protocols are designed with media delivery in mind, the fact of the matter is that the Internet was built on HTTP and optimized for HTTP delivery. Consequently this begged the question, "Why not adapt media delivery to the Internet instead of trying to adapt the entire Internet to streaming protocols?"

Move Networks proved on several occasions in 2008 that HTTP-based media delivery could be done successfully on a large scale—both on-demand and live, such as with the broadcast of the Democratic National Convention , using Microsoft® Silverlight™ as the client framework. Microsoft demonstrated this again during coverage of the 2008 Beijing Summer Olympic Games, when it prototyped its own HTTP-based adaptive streaming platform.

Content Delivery Techniques

Media delivery on the Web today uses three general delivery methods: traditional streaming, progressive download, and adaptive streaming.

Traditional Streaming

RTSP (Real-Time Streaming Protocol) is a good example of a traditional streaming protocol. RTSP is defined as a *stateful* protocol, which means that from the first time a client connects to the streaming server until the time it disconnects from the streaming server, the server keeps track of the client's state. The client communicates its state to the server by issuing it commands such as PLAY, PAUSE or TEARDOWN (the first two are obvious; the last one is used to disconnect from the server and close the streaming session).

After a session between the client and the server has been established, the server begins sending the media as a steady stream of small packets (the format of these packets is known as RTP). The size of a typical RTP packet is 1452 bytes, which means that in a video stream encoded at 1 megabits per second (Mbps), each packet carries approximately 11 milliseconds of video. In RTSP the packets can be transmitted over either UDP or TCP transports—the latter is preferred when firewalls or proxies block UDP packets, but can also lead to increased latency (TCP packets are re-sent until received).

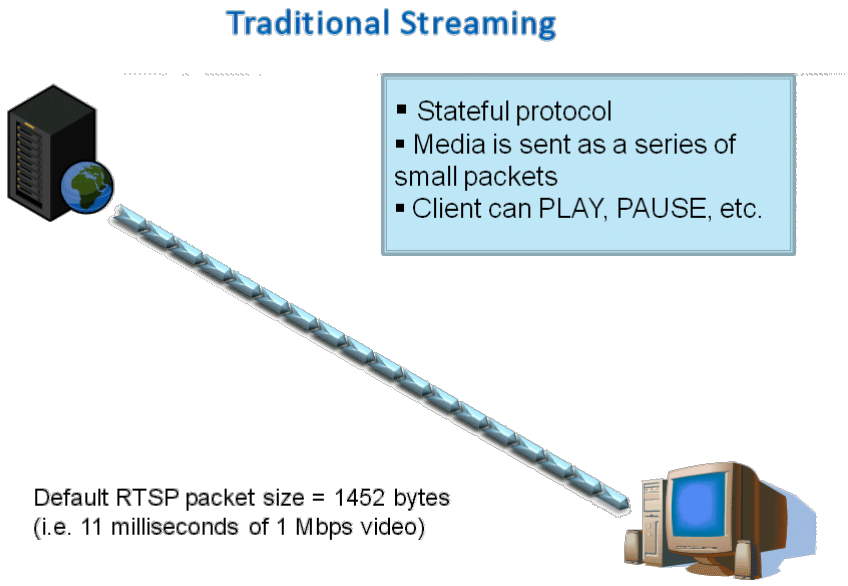

Figure 1. *RTSP is an example of a traditional streaming protocol.*

HTTP, on the other hand, is known as a *stateless* protocol. If an HTTP client requests some data, the server responds by sending the data, but it won't remember the client or its state. Each HTTP request is handled as a completely standalone one-time session.

Windows Media Services supports streaming over both RTSP and HTTP. But if HTTP is a stateless protocol, how can it be used for streaming? Windows Media Services uses a modified version of HTTP officially known as MS-WMSP (known in Windows Media Services as the Windows Media HTTP Streaming Protocol, or more commonly just as Windows Media HTTP). MS-WMSP uses standard HTTP for transfer of data and messages but also maintains session states, effectively turning it into a streaming protocol like RTSP. Windows Media Services has also supported RTSP streaming since 2003 (in Windows Media Services 9 Series) over both UDP and TCP. Its implementation of the protocol is publicly documented as MS-RTSP.

Silverlight only supports HTTP-based delivery from Windows Media Services.

The most important things to remember about traditional streaming protocols such as RTSP and Windows Media HTTP (MS-WMSP) are:

- The server sends the data packets to the client at a real-time rate only—that is, the bit rate at which the media is encoded. For example, a video encoded at 500 kilobits per second (kbps) is streamed to clients at approximately 500 kbps.
- The server only sends ahead enough data packets to fill the client buffer. The client buffer is typically between 1 and 10 seconds (Windows Media Player and Silverlight default buffer length is 5 seconds). This means that if you pause a streamed video and wait 10 minutes, still only approximately 5 seconds of video will have downloaded to the client in that time.

Other examples of traditional streaming protocols include Adobe Systems' proprietary Real Time Messaging Protocol (RTMP) and RealNetworks' RTSP over Real Data Transport (RDT) protocol. The Dynamic Streaming stream-switching feature in the Adobe® Flash® Platform is based on the RTMP protocol and is, therefore, considered a traditional streaming method—not adaptive streaming.

Progressive Download

Another common form of media delivery on the Web today is progressive download, which is nothing more than a simple file download from an HTTP Web server. Progressive download is supported by most media players and platforms, including Adobe Flash, Silverlight, and Windows Media Player. The term "progressive" stems from the fact that most player clients allow the media file to be played back while the download is still in progress—before the entire file has been fully written to disk (typically to the Web browser cache). Clients that support the HTTP 1.1 specification can also seek to positions in the media file that haven't been downloaded yet by performing byte range requests to the Web server (assuming that it also supports HTTP 1.1).

Popular video sharing Web sites on the Web today, including YouTube, Vimeo, MySpace, and MSN Soapbox, almost exclusively use progressive download.

Unlike streaming servers that rarely send more than 10 seconds of media data to the client at a time, HTTP Web servers keep the data flowing until the download is complete. If you pause a progressively downloaded video at the beginning of playback and then wait, the entire video will eventually have downloaded to your browser cache, allowing you to smoothly play the whole video without any hiccups.

There is a downside to this behavior as well—if 30 seconds into a fully downloaded 10 minute video, you decide that you don't like it and quit the video, both you and your content provider have just wasted 9 minutes and 30 seconds worth of bandwidth. To try to mitigate this problem, IIS 7.0 provides a cool extension called Bit Rate Throttling, which allows content providers to throttle the download bit rate in exactly the same way that a streaming server would to reduce costs.

HTTP-Based Adaptive Streaming

Adaptive streaming is a hybrid delivery method that acts like streaming but is based on HTTP progressive download. It's an advanced concept that uses HTTP rather than a new protocol. Both IIS Smooth Streaming and Move Networks Adaptive Stream are examples of adaptive streaming. Even though the two technologies use different codecs, formats, and encryption schemes, they both rely on HTTP as the transport protocol and perform the media download as a long series of very small progressive downloads, rather than one big progressive download.

In a typical adaptive streaming implementation, the video/audio source is cut into many short segments ("chunks") and encoded to the desired delivery format. Chunks are typically 2-to-4-seconds long. At the video codec level, this typically means that each chunk is cut along video GOP (Group of Pictures) boundaries (each chunk starts with a key frame) and has no dependencies on past or future chunks/GOPs. This allows each chunk to later be decoded independently of other chunks.

The encoded chunks are hosted on a HTTP Web server. A client requests the chunks from the Web server in a linear fashion and downloads them using plain HTTP progressive download. As the chunks are downloaded to the client, the client plays back the sequence of chunks in linear order. Because the chunks are carefully encoded without any gaps or overlaps between them, the chunks play back as a seamless video.

The "adaptive" part of the solution comes into play when the video/audio source is encoded at multiple bit rates, generating multiple chunks of various sizes for each 2-to-4-seconds of video. The client can now choose between chunks of different sizes. Because Web servers usually deliver data as fast as network bandwidth allows them to, the client can easily estimate user bandwidth and decide to download larger or smaller chunks ahead of time. The size of the playback/download buffer is fully customizable.

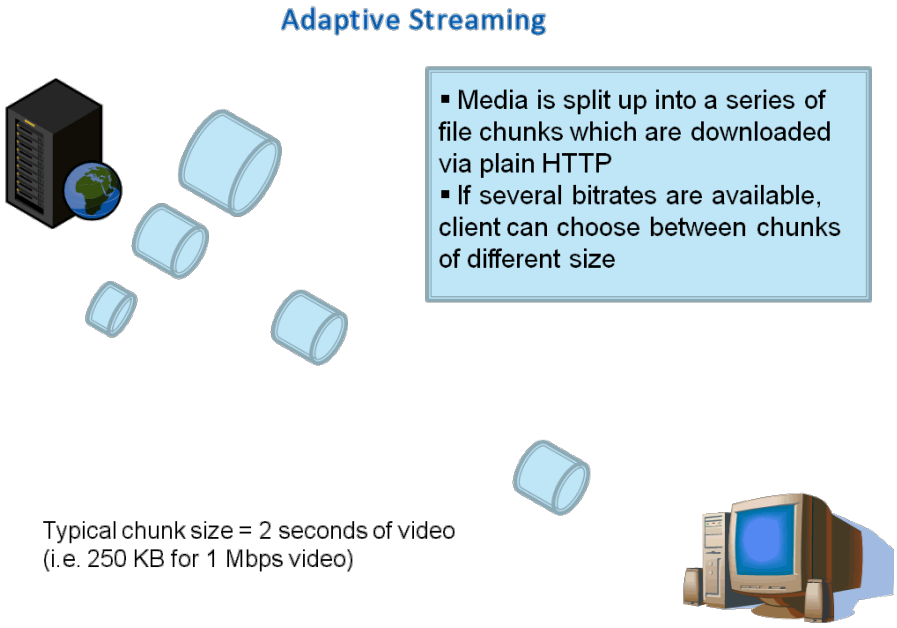

Figure 2. *Adaptive streaming is a hybrid media delivery method.*

Adaptive streaming, like other forms of HTTP delivery, offers the following advantages over traditional streaming to the content distributor:

- It's cheaper to deploy because adaptive streaming can use generic HTTP caches/proxies and doesn't require specialized servers at each node.
- It offers better scalability and reach, reducing "last mile" issues because it can dynamically adapt to inferior network conditions as it gets closer to the user's home.
- It lets the audience adapt to the content, rather than requiring content providers to guess which bit rates are most likely to be accessible to their audience.

It also offers the following benefits for the user:

- Fast start-up and seek times because start-up/seeking can be initiated on the lowest bit rate before moving to a higher bit rate.
- No buffering, no disconnects, no playback stutter (as long as the user meets the minimum bit rate requirement).
- Seamless bit rate switching based on network conditions and CPU capabilities.
- A generally consistent, **smooth** playback experience.

Microsoft created a prototype implementation of HTTP-based adaptive streaming for the NBC 2008 Beijing Summer Olympic Games Web site. To meet the project's rapid development schedule, this implementation was very straightforward. NBC used Digital Rapids and Anystream encoders to produce multiple Windows Media Video (WMV) files of different bit rates/resolutions for each source. The

encoders didn't employ any new encoding tricks but merely followed strict encoding guidelines (closed GOP, fixed-length GOP, VC-1 entry point headers, and so on.) which ensured exact frame alignment across the various bit rates of the same video. These WMV files were run through a post-processing tool that physically split each WMV file into thousands of 2-second chunks (files). The rest of the solution consisted of uploading the chunks to the CDN's Web servers and then building a Silverlight player that would download the chunks and play them in sequence.

With this implementation, NBC and Microsoft were able to offer a better-than-WMS streaming experience while using just simple HTTP download, with increased average content viewing times that directly translated to better advertising and monetization opportunities.

However, CDN operators lost many hours managing the millions of tiny files in their systems. Imagine: if each 2-seconds of video is split into a separate file and this is repeated for 5 available bit rates, you end up with 150 files for each minute of video. That's 13,500 files for a 90-minute soccer game!

So despite the NBC Olympics site being a huge success for Silverlight and HTTP-based adaptive streaming, it quickly became apparent that to productize this solution and offer improved file-management benefits, elementary design changes were required.

Introducing IIS Smooth Streaming

The IIS Media team productized the adaptive streaming solution used for the NBC Olympics site into a real server product. Its official name is **IIS Smooth Streaming**, an extension for Internet Information Services 7.0.

The IIS Media team redesigned the content creation and delivery aspects of the Olympic Games prototype to improve file management while retaining advantages from the original solution. The new design eschewed the one-file-per-chunk approach in favor of a single contiguous file for each encoded bit rate. The file format of choice would be MPEG-4.

IIS Smooth Streaming uses the MPEG-4 Part 14 (ISO/IEC 14496-12) file format as its disk (storage) and wire (transport) format. Specifically, the Smooth Streaming specification defines each chunk/GOP as an MPEG-4 Movie Fragment and stores it within a contiguous MP4 file for easy random access. One MP4 file is expected for each bit rate. When a client requests a specific source time segment from the IIS Web server, the server dynamically finds the appropriate Movie Fragment box within the contiguous MP4 file and sends it over the wire as a standalone file, thus ensuring full cacheability downstream.

In other words, with Smooth Streaming, file chunks are created virtually upon client request, but the actual video is stored on disk as a single full-length file per encoded bit rate. This offers tremendous file-management benefits.

You can see Smooth Streaming in action at http://www.smoothhd.com. The Smooth Streaming extension for IIS 7.0 is available from The Official Microsoft IIS Web site.

On the content-creation end, encoding on-demand Smooth Streaming-compatible video is already possible with Expression Encoder 2 Service Pack 1 (SP1). Note that you'll need to purchase the full version of Expression Encoder 2 to get Smooth Streaming encoding support—it's not included in the "Express" trial version. We recommend the Smooth Streaming Multi Bit Rate Calculator as a helper tool for encoding to multiple bit rate formats such as Smooth Streaming.

In addition, Microsoft is working with a number of encoding independent software vendors (ISVs) to enable support for the Smooth Streaming format in their professional encoding products.

Smooth Streaming Playback with Silverlight

Silverlight 2 already supports playback of Smooth Streaming sources. But how does it do it? Silverlight support for Smooth Streaming, including the parsing of the MPEG-4 file format, the HTTP download, the bit rate switching heuristics, etc., is provided in .NET code. This allows developers to modify and fine-tune the client adaptive streaming code as needed, instead of waiting for the next Silverlight release.

The most challenging part of Smooth Streaming Silverlight client development is the heuristics module that determines when and how to switch bit rates. Elementary stream switching functionality requires the ability to swiftly adapt to changing network conditions while not falling too far behind, but that's often not enough to deliver a great experience. One must also consider:

- What if the user has enough bandwidth but doesn't have enough CPU power to consume the high bit rates/resolutions?
- What happens when the video is paused or hidden in the background (minimized browser window)?
- What if the resolution of the best available video stream is actually larger than the screen resolution, thus wasting bandwidth?
- How large should the download buffer window be?
- How does one ensure seamless rollover to new media assets such as advertisements?

As any Web application developer will tell you, there's much more to building a good player than just setting a source URL for the media element.

Fortunately for those who prefer not to write such code from scratch, there are two options already available for adding Smooth Streaming support to your Silverlight application:

- **Expression Encoder 2 SP1 templates**. The Silverlight 2 player templates included with Expression® Encoder 2 SP1 include a ready-to-use Smooth Streaming module and complete source code (which, if needed, allows fine-tuning of the switching heuristics to adjust to particular needs of a given network or device). The Smooth Streaming object (AdaptiveStreaming.dll) can be easily integrated into any Silverlight project. See James Clarke's Weblog for additional Expression Encoder tips & tricks.
- **Open Video Player (OVP)**. The Akamai-led Open Video Player Initiative is an open-source community project that strives to provide a best-of-breed video player platform for Silverlight and Flash. The Silverlight version of the Open Video Player provides integrated support for

Smooth Streaming playback, and is the video player used by Akamai on SmoothHD.com and many of their customer sites.

Smooth Streaming Architecture

Introducing the Smooth Streaming Format

Smooth Streaming is the first Microsoft media format in over a decade to use a file format other than ASF. It's based on the ISO/IEC 14496-12 ISO Base Media File Format specification, better known as the MP4 file specification. Why MP4 and not ASF? Well, there are several reasons:

- MP4 is a lightweight container format with less overhead than ASF.
- MP4 is easier to parse in managed (.NET) code than ASF.
- MP4 is based on a widely used standard, making 3rd party adoption and support more straightforward.
- MP4 is architected with H.264 video codec support in mind. H.264 is an industry leading video compression standard that has been adopted across a broad range of Microsoft products, including Silverlight 3, Windows 7, Xbox 360, Zune and MediaRoom.
- MP4 is designed to natively support payload fragmentation within the file.

There are actually two parts to the Smooth Streaming format: the wire format, and the disk file format. In Smooth Streaming, a video is recorded in full length to the disk as a single file (one file per encoded bit rate), but it's transferred to the client as a series of small file chunks. The wire format defines the structure of the chunks that are sent by IIS to the client, whereas the file format defines the structure of the contiguous file on disk, enabling better file management. Fortunately, the MP4 specification allows MP4 to be internally organized as a series of fragments, which means that in Smooth Streaming the wire format is a direct subset of the file format.

Smooth Streaming Disk File Format

The basic unit of an MP4 file is called a "box." These boxes can contain both data and metadata. The MP4 specification allows for various ways to organize data and metadata boxes within a file. In most media scenarios, it's considered useful to have the metadata written before the data so that a player client application can have more information about the video/audio that it's about to play before it plays it. However, in live streaming scenarios it's often not possible to write the metadata up-front about the whole data stream because it's simply not yet fully known. Furthermore, less up-front metadata means less overhead, which can lead to shorter startup times. For these reasons the MP4 ISO Base Media File Format specification is designed to allow MP4 boxes to be organized in a fragmented manner, where the file can be written "as you go" as a series of short metadata/data box pairs, rather than one long metadata/data pair. The Smooth Streaming file format heavily leverages this aspect of the MP4 file specification, to the point where at Microsoft we often interchangeably refer to Smooth Streaming files as "Fragmented MP4 files" or "(f)MP4."

The following figure is a high-level overview of what a Smooth Streaming file looks like on the inside:

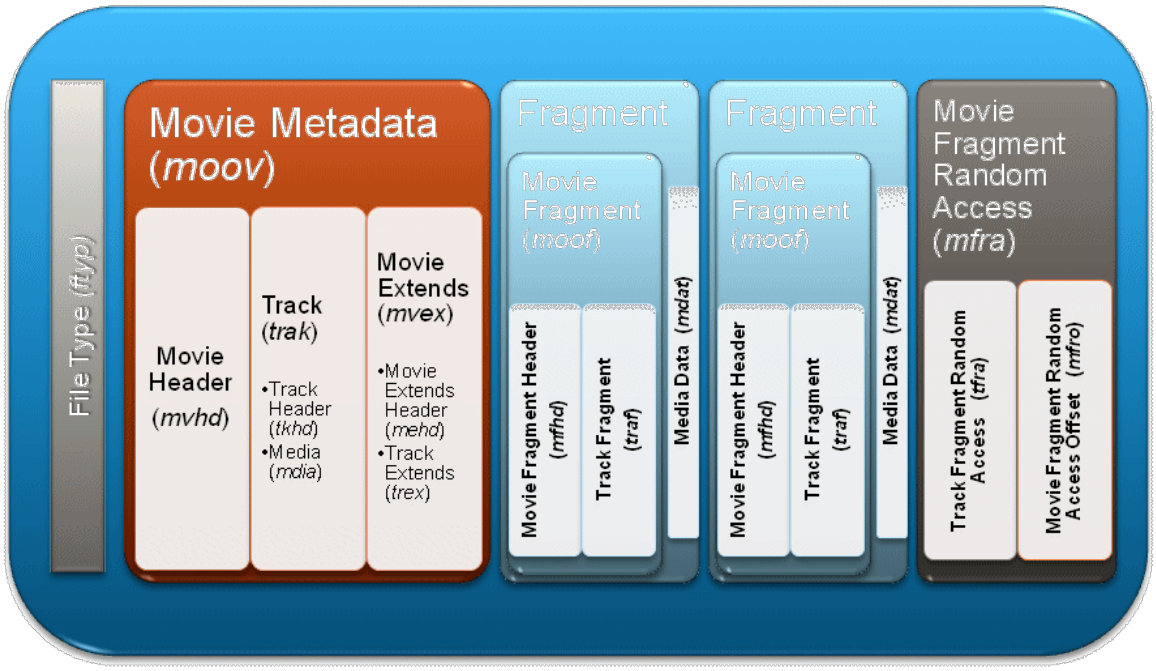

Figure 3. *Smooth Streaming File Format*

In a nutshell, the file starts with file-level metadata ('moov') that generically describes the file, but the bulk of the payload is actually contained in the fragment boxes that also carry more accurate fragment-level metadata ('moof') and media data ('mdat'). (The diagram in Figure 3 only shows 2 fragments, but a typical Smooth Streaming file has a fragment for each 2 seconds of video/audio.) Closing the file is an 'mfra' index box that allows easy and accurate seeking within the file.

Smooth Streaming Wire File Format

When a player client requests a video time slice from the IIS Web server, the server seeks to the appropriate starting fragment in the MP4 file and then lifts the fragment out of the file and sends it over the wire to the client. This is why we refer to the fragments as the "wire format." This technique greatly enhances the efficiency of the IIS Web server because it doesn't induce any re-muxing or rewriting overhead.

The following figure shows what an MP4 fragment looks like in more detail:

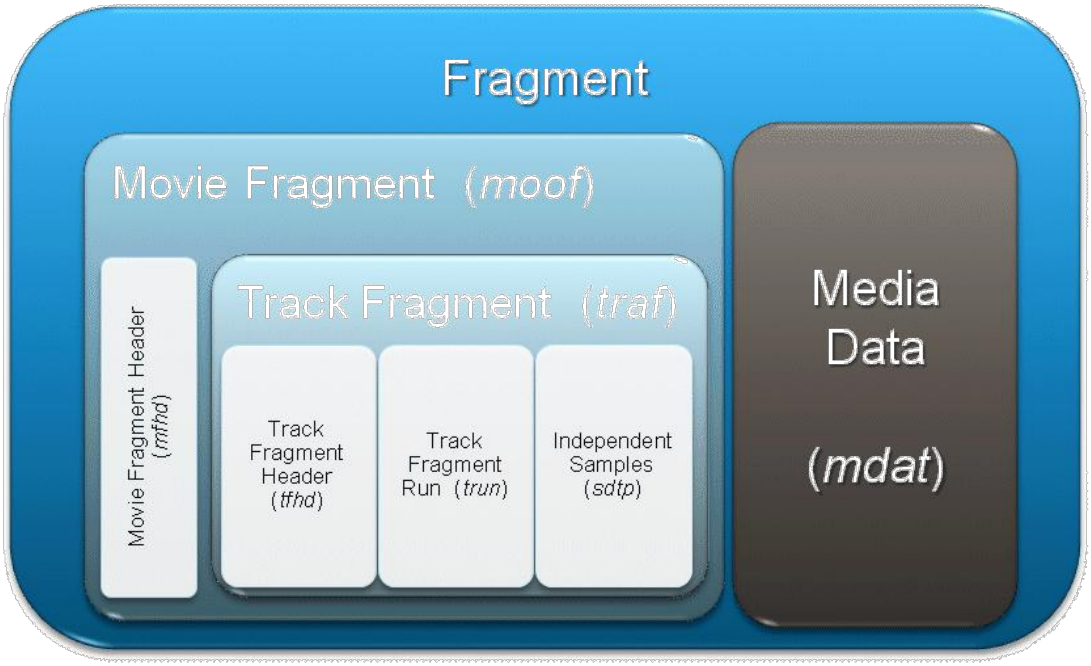

Figure 4. *Smooth Streaming Wire Format.*

Within the guidelines of the MP4 ISO Base Media File Format specification, the Smooth Streaming format uses a custom box organization schema and some custom boxes. To differentiate Smooth Streaming files from "vanilla" MP4 files, we use new file extensions: ***.ismv** (video+audio) and ***.isma** (audio only).

Smooth Streaming Media Assets

A typical Smooth Streaming media asset (presentation) consists of the following files:

- MP4 files containing video/audio
  - ***.ismv**
    - Contains video and audio, or only video
    - 1 ISMV file per encoded video bit rate
  - ***.isma**
    - Contains only audio
    - In videos with audio, the audio track can be muxed into an ISMV file instead of a separate ISMA file
- Server manifest file
  - ***.ism**
    - Describes the relationships between the media tracks, bit rates and files on disk
    - Only used by the IIS Smooth Streaming server—not by clients
- Client manifest file
  - ***.ismc**

- Describes the available streams to the client: the codecs used, bit rates encoded, video resolutions, markers, captions, etc.
- It's the first file delivered to the client

The manifest files are XML-formatted files. The server manifest file format is based specifically on the SMIL 2.0 XML format specification.

A folder containing a single Smooth Streaming presentation might look something like the following:

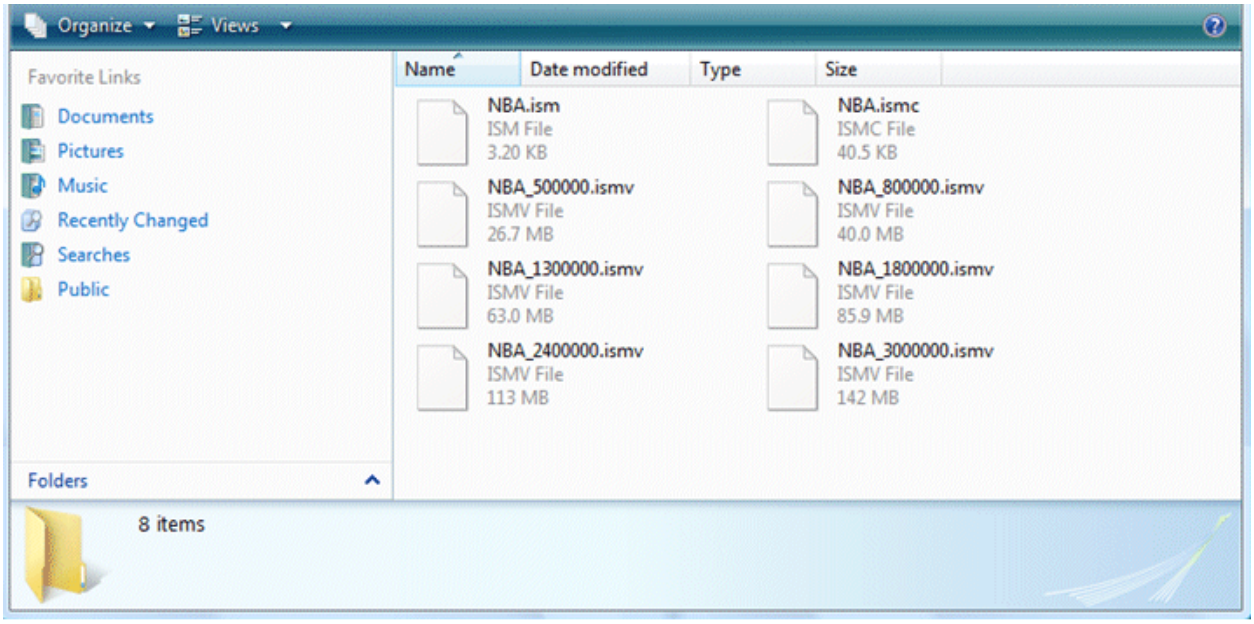

Figure 5. *A folder containing a Smooth Streaming encoded presentation. In this particular case, the audio track is contained in the NBA_3000000.ismv file.*

Smooth Streaming Manifest Files

The Smooth Streaming Wire/File Format specification defines the manifest XML language as well as the MP4 box structure. Because the manifests are based on XML, they are highly extensible. Among the features already included in the current Smooth Streaming format specification is support for the following:

- VC-1, WMA, H.264 and AAC codecs
- Text streams
- Multi-language audio tracks
- Alternate video and audio tracks (for example, multiple camera angles, director's commentary, etc.)
- Multiple hardware profiles (for example, a bit rate targeted at different playback devices)
- Script commands, markers/chapters, captions
- Client manifest Gzip compression

- URL obfuscation
- Live encoding and streaming

Sample Smooth Streaming on-demand server manifest files (.ism) and Smooth Streaming client manifest files (.ismc) are included in the IIS Smooth Streaming Beta Sample Content for IIS Smooth Streaming Beta.

Smooth Streaming Playback

Microsoft's adaptive streaming prototype (used for NBC Olympics 2008) relied on physically chopping up long video files into small file chunks. To retrieve the chunks for the Web server, the player client simply needed to download the files in a logical sequence: 00001.vid, 00002.vid, 00003.vid, and so on.

Smooth Streaming uses a more sophisticated file format and server design. The videos are no longer split up into thousands of file chunks, but are instead "virtually" split into fragments (typically one fragment per video GOP) and stored within a single contiguous MP4 file. This implies two significant changes in server and client design:

- The server must translate URL requests into exact byte range offsets within the MP4 file
- The client can request chunks in a more developer-friendly manner (for example, by timecode instead of by index number)

The first thing a player client requests from the Smooth Streaming server is the ***.ismc** client manifest. The manifest tells it which codecs were used to compress the content (so that the client runtime can initialize the correct decoder and build the playback pipeline), which bit rates and resolutions are available, and a list of the available chunks (with either their start times or durations).

With IIS Smooth Streaming, clients request fragments in the form of a RESTful URL:

- http://video.foo.com/NBA.ism/QualityLevels(400000)/Fragments(video=610275114)

The values passed in the URL represent encoded bit rate (400000) and the fragment start offset (610275114) expressed in an agreed-upon time unit (usually 100 nanoseconds (ns)). These values are known from the client manifest.

After receiving the client request, IIS Smooth Streaming looks up the quality level (bit rate) in the corresponding *.ism server manifest and maps it to a physical *.ismv or *.isma file on disk. It then reads the appropriate MP4 file, and based on its 'tfra' index box, figures out which fragment box ('moof' + 'mdat') corresponds to the requested start time offset. It then extracts the fragment box and sends it over the wire to the client as a standalone file. This is a particularly important part of the overall design because the sent fragment/file can now be automatically cached further down the network, potentially saving the origin server from sending the same fragment/file again to another client that requests the same RESTful URL.

Requesting chunks of video/audio from the server is easy. But what about the bit-rate switching that makes adaptive streaming so effective? This part of the Smooth Streaming experience is implemented

entirely in client-side Silverlight application code—the server plays no part in the bit-rate switching process. The client-side code looks at chunk download times, buffer fullness, rendered frame rates, and other factors, and decides when to request higher or lower bit rates from the server. Remember, if during the encoding process we ensure that all bit rates of the same source are perfectly frame-aligned (same length GOPs, no dropped frames, etc.), then switching between bit rates is completely seamless.

Summary

Smooth Streaming is Microsoft's implementation of HTTP-based adaptive streaming, which is a hybrid media delivery method. It acts like streaming, but is based on HTTP progressive download. The HTTP downloads are performed in a series of small chunks, allowing the media to be easily and cheaply cached along the edge of the network, closer to clients. Providing multiple encoded bit rates of the same media source also allows clients to seamlessly and dynamically switch between bit rates depending on network conditions and CPU power. The resulting user experience is one of reliable, consistent playback without stutter, buffering or "last mile" congestion. In one word: **Smooth**.

For More Information

Media on IIS 7.0	http://iis.net/media
Microsoft Silverlight—Media	http://www.microsoft.com/silverlight/overview/media.aspx
Microsoft Expression	http://www.microsoft.com/expression
Silverlight Developer Center	http://msdn.microsoft.com/silverlight
Microsoft Silverlight Community	http://silverlight.net/community/
Windows Media Services 2008	http://www.microsoft.com/windows/windowsmedia/forpros/serve/prodinfo2008.aspx

Legal Notice

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.