Watching Video over the Web, Part I

Streaming Protocols

Ali C. Begen, Tankut Akgul and Mark Baugher, Cisco

A U.S. consumer watches TV for about five hours a day on average. While the majority of viewed content is still the broadcast TV programming, the share of the time-shifted content has been ever increasing. One third of the U.S. consumers currently use a digital video recorder (DVR)-like device for time-shifting, however, the trends are showing that more and more consumers are going to the Web to watch their favorite shows and movies on a computer or mobile device. Increasingly, the Web is coming to the digital TV, which incorporates movie downloads and streaming using Web protocols. In this first part of a two-part article, the authors describe both conventional and emerging streaming solutions using Web and non-Web protocols and provide a detailed comparison.

Summer 2010, time for the 19th FIFA World Cup. It was an exhilarating whole month with total 64 matches played by 32 national teams. It is no surprise that the World Cup has been one of the most watched events worldwide throughout history. This year, there was even more interest as it was the first time in the history of the tournament that some 25 games were broadcast in 3D. But more importantly, many more viewers than ever watched the games over the Web. With the recent developments in video streaming technologies and increase in the broadband Internet access penetration, fans enjoyed watching the games in HD or near-HD quality on their computers, smartphones and other connected devices including TVs. In the U.S., 45% of the daily World Cup TV audience watched the matches in a non-home environment or on a non-TV platform[1]. ESPN3.com reached over seven million unique viewers and delivered over 15 million hours of content.

Just a few months ago, we witnessed the same phenomenon in the Vancouver 2010 Winter Games. In Canada where the population is about 34 million, there were almost four million unique viewers who watched the games on the Internet. CTV, the national TV channel that broadcast the games, made 300 events available on the Web. Canadian Internet users consumed a total of 6.3 million hours of live and 0.9 million hours of on-demand content, resulting in a total 6.2 petabytes delivered in about two weeks[2]. Two years ago, the Beijing Summer Games also attracted many online viewers - both for live and on-demand viewing. In the U.S., NBC delivered more than 1100 years of video content to about 52 million unique online viewers during the games.

[1] http://www.espnmediazone3.com/us/2010/07/espn-xp-world-cup-dispatch-4-through-742010/
[2] http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=4000007347

The sport events are not the sole type of content attractive to online viewers, though. In the past few years, many content providers have started making their regular and premium content such as news, series, shows and movies available on their Web sites. Despite the certain geographical restrictions on many of these Web sites, the consumers saw a great proliferation in the amount of content they had access to. Some content providers or TV channels actually did not impose any restrictions on the viewer location, and thus were transformed from being a regional TV channel to become a global one, extending their reach for ad revenues in an unforeseen manner.

The consumers' desire for being able to access practically limitless amount of content any time they wanted to and the drop in the delivery costs further hastened the deployment of streaming services. New Web sites that do content aggregation such as Hulu came alive. In May 2010, Hulu had more than 40 million unique viewers in the U.S., streaming more than one billion videos a month[3]. Remarkably, these numbers have been steadily increasing. Netflix, who is the largest subscription service for DVD rental and streaming video, currently has 13 million members in the U.S., 55% of whom use its streaming services on a variety of devices[4].

Internet video, also known as over-the-top (OTT) services, can be divided into distinct categories of (*i*) user-generated content from mostly amateurs (e.g., majority of the content served by YouTube), (*ii*) professionally generated content from studios and networks to promote their commercial offerings and programming (e.g., the content served by ABC.com, hulu.com), and (*iii*) direct movie sales to consumers over the Internet (also referred to as Electronic Sell-Through – EST). In the last category, Netflix, Apple TV and new undertakings such as Ultraviolet will greatly increase the amount of video offerings on the Internet.

Cisco's Visual Networking Index (VNI)[5] suggests that traffic volumes in the order of tens and hundreds of exabytes (one billion gigabytes) and zettabytes (one thousand exabytes) are not that remote. Over the next few years, about 90% of the bits carried in the Internet will be video related and consumed by over a billion users. While some portion of these video bits will be for managed services such as cable TV and IPTV, we cannot ignore the amount of bits for the unmanaged (OTT) services. Actually, by the end of 2010, the Internet video traffic will surpass peer-to-peer file sharing for the first time ever and emerge as the top Internet traffic contributor.

Cable TV and IPTV services offered by a service provider run over managed networks for distribution since these services use multicast transport and require certain QoS features (1). In contrast, conventional streaming technologies such as Microsoft's Windows Media, Apple's

[3]http://www.comscore.com/Press_Events/Press_Releases/2010/6/comScore_Releases_May_2010_U.S._Online_Video_Rankings
[4] http://www.netflix.com/MediaCenter?id=5379
[5] http://www.cisco.com/web/go/vni

QuickTime and Adobe's Flash as well as the emerging adaptive streaming technologies such as Microsoft's Smooth Streaming, Apple's HTTP Live Streaming and Adobe's HTTP Dynamic Streaming run over mostly unmanaged, i.e., best-effort, networks. These streaming technologies send the content to the viewer over a unicast connection (from a server or content delivery network (CDN) facilitating content distribution) through either a proprietary streaming protocol running on top of an existing transport protocol, mostly TCP and occasionally UDP, or the standard HTTP protocol that is over TCP.

Historically, progressive download, which uses HTTP over TCP, has been quite popular for online content viewing due to its simplicity. In progressive download, the playout can start as soon as enough and necessary data is retrieved and buffered. Today, YouTube delivers over two billion videos a day with this approach. However, progressive download does not offer the flexibility and rich features of streaming. Before the download starts, the viewer needs to choose the most appropriate version if there are multiple offerings with different resolutions and qualities of the same content. If there is not enough bandwidth for the selected version, the viewer may experience frequent freezes and re-buffering. The trick modes such as fast-forward seek/play or rewind, are often unavailable or limited. These limitations are likely to inhibit the growth of large volume (including HD) movie distribution on the Internet. A new approach, which we refer to as Adaptive Streaming, is emerging to address these shortcomings while preserving the simplicity of progressive download.

Adaptive streaming is a hybrid of progressive download and streaming. On one hand, it is pull-based as is progressive download: the adaptive streaming client sends HTTP request messages to retrieve particular segments of the content from an HTTP server, and then renders the media while the file is being transferred. On the other hand, these segments are short duration, enabling the client to download only whatever is necessary and employ trick modes much more efficiently, which gives the impression that the client is streaming. However, more importantly, the short-duration segments (e.g., MPEG4 file fragments) are available at multiple bitrates, corresponding to different resolutions and quality levels, and the client may switch between different bitrates at each request. The client strives to always retrieve the next best segment after examining a variety of parameters related to the available network resources such as available bandwidth and the state of the TCP connection(s), device capabilities such as display resolution and available CPU, and current streaming conditions such as size of the playback buffer. The goal is to provide the best quality of experience by enabling display of the highest achievable quality, faster start-up, quicker seeking, and reducing skips, freezes and stutters.

As adaptive streaming uses HTTP, it benefits from the ubiquitous connectivity that HTTP has to offer. Today, any connected device supports HTTP in some form. Since it is a pull-based protocol that easily traverses firewalls and NAT devices, it keeps minimal state information on the server side, which makes HTTP servers potentially more scalable than conventional push-based streaming

servers. To the existing HTTP caching infrastructure, adaptive streaming is no different than any other HTTP application. Individual segments of any content are separately cacheable as regular Web objects using HTTP or any RESTful (conforming to the Representational State Transfer constraints) Web protocol. This allows distributed CDNs to greatly enhance the scalability of content distribution.

In this first installment of a two-part series, we describe several streaming, both push and pull-based, solutions that exist today, motivate the need for adaptive streaming, and provide a detailed comparison between different methods.

Media Streaming

Transmission of content between different nodes on a network can be performed in a variety of ways. The type of the content being transferred and the underlying network conditions usually determine the methods used for communication. In case of a simple file transfer over a lossy network, the emphasis is on reliable delivery. The packets can be protected against losses with added redundancy or the lost packets can be recovered by retransmissions. When it comes to audio/video media delivery with real-time viewing requirements, the emphasis is on low latency and efficient transmission in order to enable the best possible viewing experience. Occasional losses may be tolerated in this case. The structure of the packets and the algorithms used for real-time media transmission on a given network collectively define the media streaming protocol. Although various media streaming protocols available today differ in implementation details, they can be classified into two main categories: push-based protocols and pull-based protocols.

Push-Based Media Streaming Protocols

In push-based streaming protocols, once a connection is established between a server and a client, the server is always on and streams packets to the client until the session is torn down or interrupted by the client. Consequently, in push-based streaming, the server maintains a connection state with the client and listens for commands sent by the client regarding session state changes. Real-time Streaming Protocol (RTSP), specified in RFC 2326[6], is one of the most common session control protocols used in push-based streaming.

Push-based streaming protocols generally utilize Real-time Transport Protocol (RTP), specified in RFC 3550, or equivalent packet formats for data transmission. RTP usually runs on User Datagram Protocol (UDP), which is a stateless protocol without any inherent rate-control mechanisms. This allows the server to push packets to the client at a bitrate that is dependent on the client/server

[6] All RFC documents are freely accessible from http://www.ietf.org/rfc.html.

implementation at the application level rather than the underlying transport protocol. This makes RTP a nice fit for low-latency and best-effort media transmission.

In conventional push-based streaming, the content is usually transmitted at the media encoding bitrate to match the client's media consumption rate. In normal circumstances, this ensures that the client buffer levels stay stable over time. It also optimizes the use of network resources because the client usually cannot consume at any rate above the encoding bitrate; consequently, transmitting above the encoding bitrate would unnecessarily load the network. Moreover, this may not be even possible for live streams where the stream is being encoded on-the-fly. However, if packet loss or transmission delays occur over the network, the client's packet retrieval rate may drop below its consumption rate and the client's buffer may start draining, which may eventually result in a buffer underflow and interrupt the media playback. This is where bitrate adaptation comes into play.

To prevent a buffer underflow, the media server may dynamically switch to a lower bitrate stream. This, in turn, reduces the media consumption rate at the client-side and therefore counteracts the effect of network bandwidth capacity loss. Since a sudden drop in encoding bitrate may result in noticeable visual quality degradation, the encoding bitrate reduction may be performed in multiple intermediate steps until the client's consumption rate matches or drops below its available receive bandwidth. When the network conditions improve, the server does the opposite and switches to a higher bitrate stream instead. Again, this process may be performed in multiple intermediate steps not to cause a sudden overload on the network. Provided that the stream is not a live stream and the network capacity allows for a higher bitrate transmission, the server may also choose to send over packets at a higher rate than the media encoding rate to fill up the client's buffer. By dynamically monitoring the available bandwidth and buffer levels and adjusting the transmission rate via stream switching, push-based adaptive streaming may achieve smooth playback experience at the best possible quality level without pauses or stuttering.

The bandwidth monitoring is usually performed on the client. The client computes network metrics such as round-trip time (RTT), packet loss and network jitter periodically. The client may use this information directly to make decisions for when to switch to a higher or a lower bitrate stream. Alternatively, the client may communicate this information along with its buffer levels to the server via receiver reports and let the server make those decisions. Such reports are usually transmitted via Real-time Control Protocol (RTCP).

In the following section, we study the mechanisms defined in 3rd Generation Partnership Project (3GPP) that constitute an example for how bitrate adaptation can be implemented in push-based streaming servers.

Bitrate Adaptation in 3GPP

3GPP is collaboration between telecommunications associations whose scope is to produce technical specifications for 3G mobile systems based on Global System for Mobile Communications (GSM) networks. The bitrate adaptation mechanisms defined in 3GPP standards are used by popular push-based streaming servers such as Helix streaming server by Real Networks or QuickTime Streaming Server by Apple.

In its Release 6, 3GPP defined mechanisms for the media client to inform the media server for dynamically switching the transmission bitrate (2) (3). The algorithms used to collect real-time statistics about available bandwidth and client buffer levels and adapting the bitrate accordingly are implementation specific. However, in a typical implementation, the process goes as depicted in Figure 1.

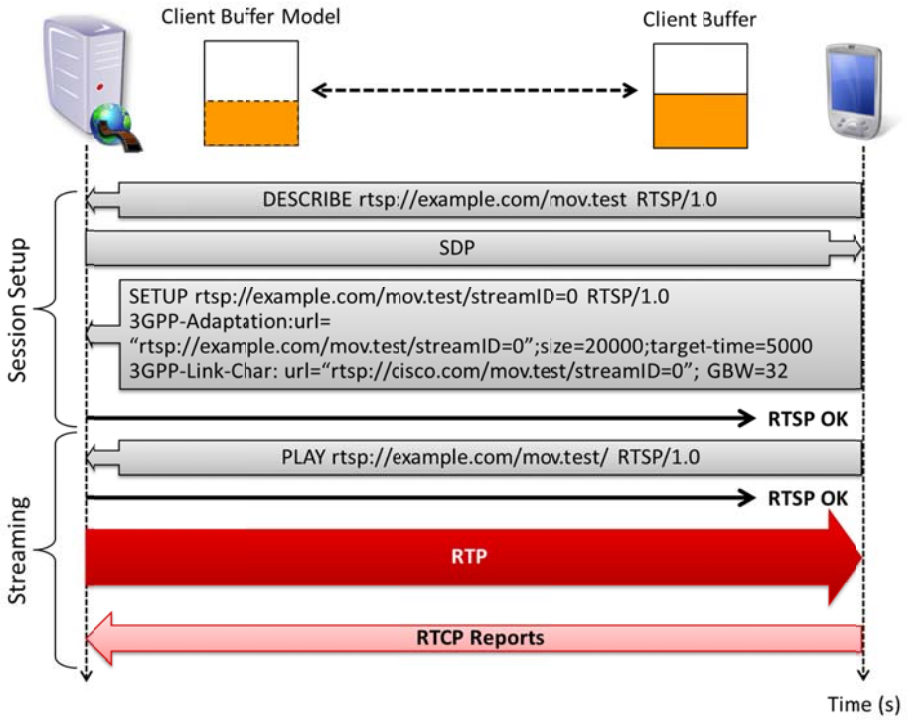

Figure 1: Example 3GPP streaming session setup with bitrate adaptation.

During session setup, the server sends a list of available streams and their properties such as bitrate and codec to the client via Session Description Protocol (SDP), specified in RFC 4566, in response to client's RTSP `describe` message. Receiving session description, the client picks from within the available audio and video streams that best fit to its link speed and decoding capabilities. Then, the client sends RTSP `setup` messages (one for each unique audio and video stream) to the

server to prepare the server for sending out the selected streams. The RTSP message header contains a `3GPP-Adaptation` parameter that informs the server about the client's buffer size (size attribute) and minimum required buffering (target-time attribute) to ensure interrupt-free playback. It also contains a `3GPP-Link-Char` parameter that provides information about client's guaranteed receive bandwidth (GBW attribute).

After the server obtains information about the client's buffer and receive bandwidth, it sets up a buffer model to simulate the changes in client's receive buffer levels. In this buffer model, the server tries to satisfy the minimum required buffer level while at the same time it tries to prevent overflows. Since the network conditions and buffering requirements may change over time, the client keeps notifying the server via periodic RTCP receiver reports. These RTCP messages carry information such as available free space in client's receive buffer, playout delay (the time difference between the presentation time of the next frame in the decoder buffer and the time the RTCP message is being sent) and estimated network bandwidth. Using this information, the server can maintain an accurate model of client's receive buffer and can make informed decisions about when to switch to a higher or a lower bitrate stream dynamically.

The upshift and downshift buffer level thresholds for the stream are usually pre-programmed on the server. For instance, if the server detects that the client's buffer level is below the next downshift threshold, it switches to the next lower encoding rate available to prevent the buffer from draining any further. On the other hand, if the buffer level exceeds the next upshift threshold, the server switches to the next higher encoding rate provided that the network bandwidth can support that rate. If not, the server may choose to slow down the transmission rate to prevent a buffer overflow.

Pull-Based Media Streaming Protocols

In pull-based streaming protocols, the media client is the active entity that requests the content from the media server. Therefore, the server response depends on the client's requests where the server is otherwise idle or blocked for that client. Consequently, the bitrate at which the client receives the content is dependent upon the client and the available network bandwidth. As the primary download protocol of the Internet, HTTP is a common communication protocol that pull-based media delivery is based on.

Progressive download is one of the most widely used pull-based media streaming methods available on IP networks today. In progressive download, the media client issues an HTTP request to the server and starts pulling the content from the server as fast as possible. Once a minimum required buffer level is obtained, the client starts playing the media while at the same time it continues to download the content from the server in the background. As long as the download rate is not smaller than the playback rate, the client buffer is kept at a sufficient level to continue the

playback without any interruption. However, if the network conditions degrade, the download rate may fall behind the playback rate and an eventual buffer underflow may result.

Similar to methods used in push-based streaming, bitrate adaptation is a measure taken to prevent buffer underflow in pull-based streaming protocols. Figure 2 shows an example implementation for bitrate adaptation. The media content is divided into short-duration media segments (also called fragments) each of which is encoded at various bitrates. Each fragment can be decoded independently. When the fragments are played back to back, the original media stream can be reconstructed seamlessly. During download, the media client dynamically picks the fragment with the right encoding bitrate that matches or is below the available bandwidth and requests that fragment from the media server. This way the client can adapt its media consumption rate according to the available receive bandwidth.

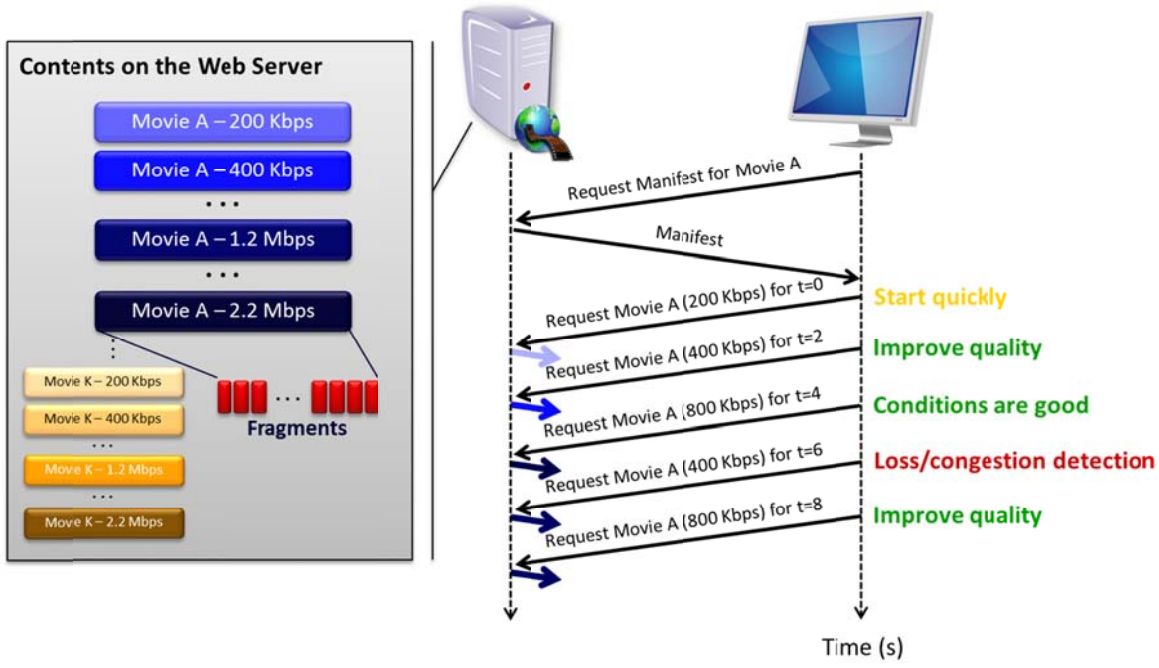

Figure 2: Example bitrate adaptation in pull-based adaptive streaming.

Although the structure of the media fragments differs among implementations, the basic principle for fragment construction is the same. In the case where audio and video are not interleaved, each audio frame usually consists of constant duration audio samples in the milliseconds range and each frame is usually decodable on its own for common audio codecs. Therefore, audio data can easily be stuffed into a media fragment by combining sufficient number of audio frames to match the fragment duration. For video, on the other hand, the frames are not necessarily decodable independently due to common temporal prediction applied between the frames. Therefore, for video, the partitioning for fragment construction is performed at the Group of Pictures (GoP)

boundary instead. In video coding, a GoP is a sequence of frames that start with an intra-coded frame (I-frame) that can be decoded independently followed by predicted frames (P-/B-frames) that depend on other frames. If the predicted frames within a GoP depend only on the frames within the same GoP, the GoP is called a *closed GoP*. Otherwise, the GoP is called an *open GoP*. Since, a closed GoP is self-contained (i.e., it can be decoded independent of other GoPs), one can construct a fragment from a closed GoP straightforwardly. During encoding, the number of frames in a GoP can be adjusted such that the total duration of the frames in the GoP is equal to the desired fragment duration. However, if the fragment duration is large compared to a typical GoP size, then it is preferable to pack more than one GoP into a fragment.

In the following sections, we look into two different implementations of pull-based adaptive streaming protocols that are based on multi-bitrate fragments. These are Smooth Streaming by Microsoft and HTTP Live Streaming by Apple.

Microsoft Smooth Streaming

Microsoft's Smooth Streaming implementation is based on Protected Interoperable File Format (PIFF) (4), which is an extension to MPEG4 (MP4) specification (5). In Smooth Streaming implementation, all fragments having the same bitrate are stored in a single MP4 file. Therefore, there is a separate file for each available bitrate. Fragments are usually two seconds long and typically contain a single GoP.

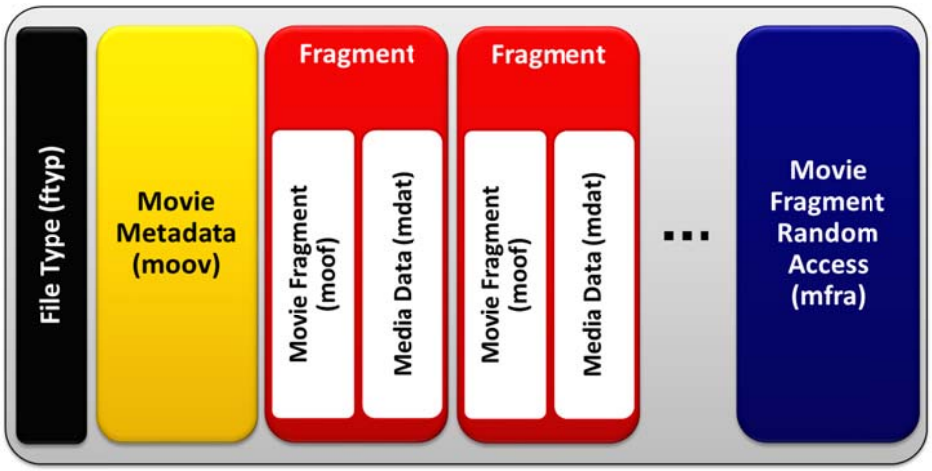

Figure 3: Fragmented MP4 file format (Reproduced from (4)).

Figure 3 shows the structure of a fragmented MP4 file. A fragmented MP4 file is composed of a hierarchical data structure. The building block of this hierarchy is called a *box* (known as an *atom* in QuickTime). There are boxes that contain audio/video data as well as boxes that contain metadata. Each box type is designated by a four-letter identifier. The file starts with an ftyp box that describes

the version information for the specification(s) the file complies with. It is followed by a moov box that describes the media tracks available in the file. The audio/video media data for a single fragment is contained within a box of type mdat. In a fragmented MP4 container structure (5), an mdat box is immediately preceded by a box of type moof that contains metadata for that fragment. The moof box may contain metadata and signaling information such as a sequence counter for the fragment, the number of samples within the fragment and duration of each sample.

In order for the client to request a fragment with a specific bitrate and start time, it first needs to know whether that fragment is available on the server. This information is communicated to the client at the beginning of the session via a client-side manifest file. In addition to bitrates, the client-side manifest file describes codecs, video resolutions, captions and other auxiliary information for the available streams.

```
GET /sample/v_720p.ism/QualityLevels(1500000)/Fragments(video=160577243) HTTP/1.1
```

Figure 4: An example HTTP request for a downloading a fragment.

Once the client has downloaded the manifest file, it starts making HTTP GET requests to the server for the individual fragments. Each fragment is downloaded via a unique HTTP request-response pair. The HTTP request message header contains two pieces of information. These are the bitrate and the time offset of the requested fragment as shown in Figure 4, which is a Microsoft Smooth Streaming Transport client request.

When the media server gets an HTTP-encapsulated request from the client for a particular media fragment, it first needs to determine which MP4 file to search to locate that fragment. This information is contained in another manifest file, which is called server-side manifest. This manifest file maps MP4 files to the bitrates of the fragments they contain. The server looks up the bitrate information it receives from the client (as seen in Figure 4) in the manifest file and finds the corresponding MP4 file to search.

After the correct file is found, the next step is to locate the requested fragment in that file. This is achieved via indexing data. As seen in Figure 3, an MP4 file also contains boxes of type mfra that contain the location and presentation time of a fragment. The media server uses the time offset information it receives from the client to find a match with a fragment index in the MP4 file. Once the server locates the fragment in the file, it sends the contained moof box followed by the mdat box, and these make up the fragment on the wire.

Apple HTTP Live Streaming

Apple's HTTP Live Streaming implementation (6) follows a different approach for fragment (referred to as media segment in the implementation) storage. HTTP Live Streaming is based on ISO/IEC 13818-1 MPEG2 Transport Stream file format. As opposed to Smooth Streaming, here each media segment is stored in a separate transport stream file. A process named *Stream Segmenter*, which is typically a software component, reads a continuous transport stream file and divides it into equal duration segments. The default segment duration is 10 seconds. This is a longer duration compared to two seconds used in Microsoft Smooth Streaming. Longer fragment duration allows better compression since it provides better temporal redundancy and at the same time reduces the number of fragments. However, it also reduces the granularity at which fragment-switching decisions can be made. Therefore, long fragment duration may not be as good in terms of adapting to dynamic bandwidth changes.

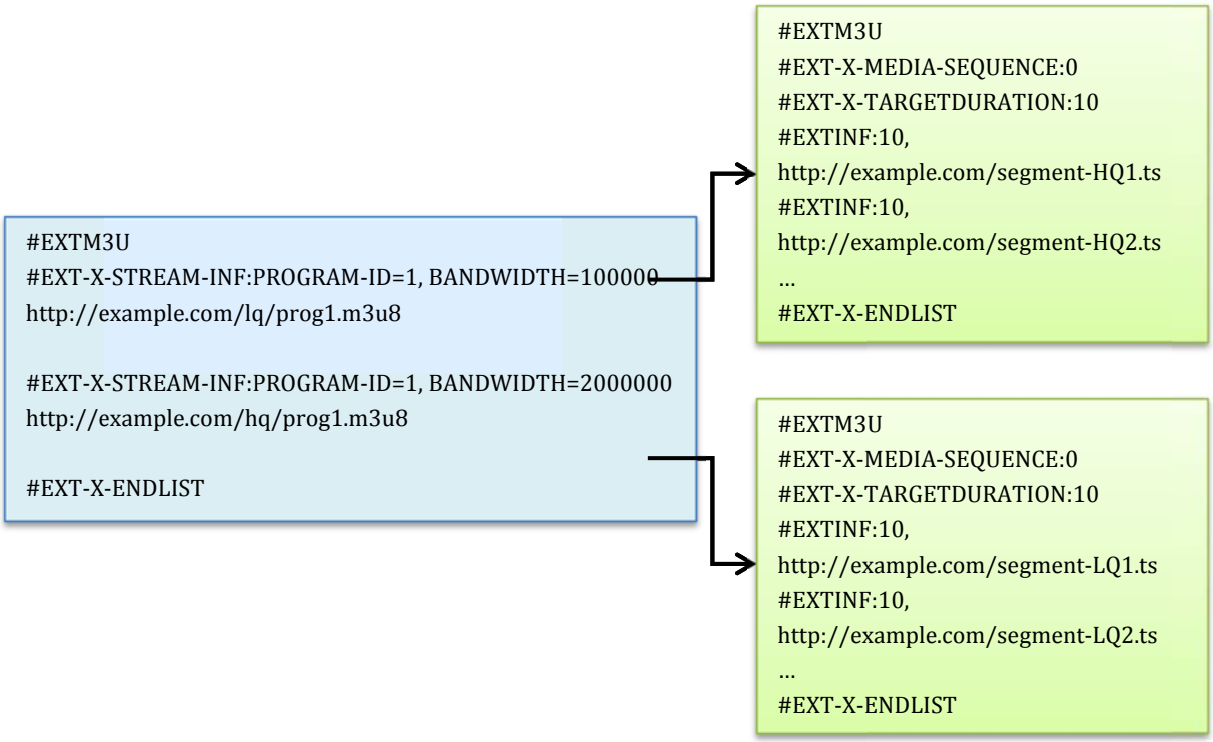

Figure 5: Example showing HTTP Live Streaming playlists.

Similar to Microsoft's Smooth Streaming implementation, a client-side manifest file keeps a record of the media segments available for the client. The manifest file format is an extension to the MP3 playlist file standard and is defined in (6). Figure 5 shows an example of the manifest files organized in a two-level hierarchy. Each file starts with an `EXTM3U` tag that distinguishes the file

from an ordinary MP3 playlist. The file on the left is a higher-level file that links to two other lower-level files on the right. The link for each lower-level file is specified in a Uniform Resource Identifier (URI) that is always preceded by an `EXT-X-STREAM-INF` tag. `EXT-X-STREAM-INF` tag has an attribute named `BANDWIDTH` that indicates that the corresponding lower-level file is a manifest for an alternate encoding of the segments it lists. In the example, the stream has two alternate encodings for each segment, that is, one low-quality (100 Kbps) encoding and one high-quality (2 Mbps) encoding.

The lower-level files on the right provide the links for the individual media segments available for download. Following the `EXTM3U` tag, is an `EXT-X-MEDIA-SEQUENCE` tag that lists the sequence number of the first segment in the playlist. Each segment in the playlist typically has a unique sequence number, and sequence numbers increment by one starting from the first segment's sequence number. Finally, the URIs for the individual segments follow each marked with an `EXTINF` tag. This tag has an attribute separated by a semicolon that indicates the duration of the corresponding media segment. In the example, each segment is 10 seconds long.

Example Functional Diagram for Client-Side Pull-Based Adaptive Streaming

A client-side pull-based adaptive streaming implementation is illustrated in Figure 6. At minimum, the server provides standard responses to HTTP GET requests. The client gets a manifest file that identifies files that contain media presentations at alternative bitrates. The client acquires media from fragments of a file over one or more connections according to the playout buffer state and other conditions.

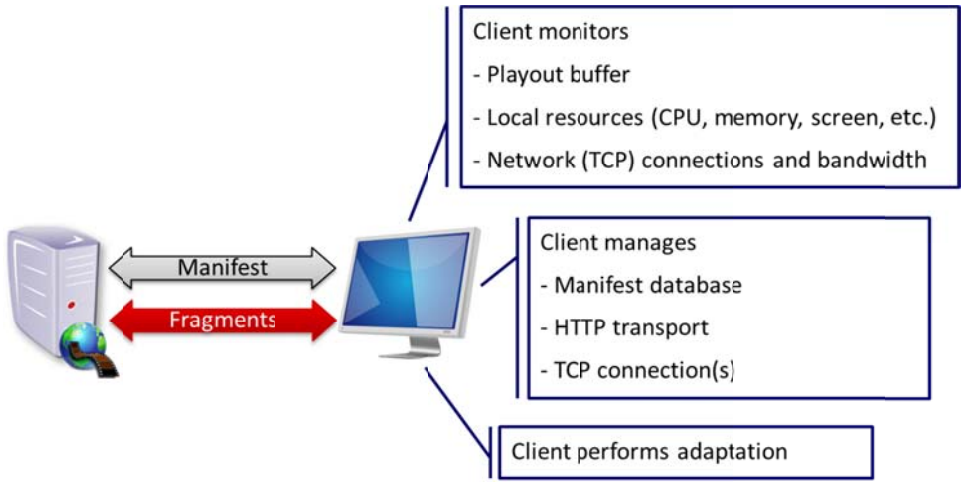

Figure 6: A client-side pull-based adaptive streaming implementation.

The basic client/server adaptive streaming configuration requires the general functions shown on the client-side of Figure 6.

- The client needs a file, called a *client-side manifest* or a *playlist* to map fragment requests to specific files or byte ranges into files. In some adaptive streaming schemes, there is a similar file on the server to translate client requests.
- Managing a playout buffer is also a basic function that is always needed on an adaptive streaming client, which uses adaptation logic to select fragments from a file at a particular bitrate in response to buffer state and potentially other variables. Typically, an adaptive streaming client keeps a playout buffer of several seconds, e.g., 5 – 30 seconds.
- A transport is needed to communicate requests from the client to the server; the pull-based adaptive streaming schemes that we survey in this article use HTTP GET requests, either to a standard HTTP Web server or to a specialized Web services API that is supported by a special service on the server (such as a Smooth Streaming Transport application running on Microsoft's Internet Information Services server).
- The GET requests use a single TCP connection by default, but some adaptive streaming implementations support using multiple concurrent TCP connections for requesting multiple fragments at the same time or for pulling audio and video in parallel.

The client is pre-configured to request a movie at a certain bitrate, or profile, based on the result of network tests or simple configuration script. When a profile is selected and the URI associated with the profile is found in the manifest, the client establishes one or more connections to the server. We have observed that different products employ different strategies. For example, some products use only one TCP connection for GET requests to a file whereas others open and close multiple TCP connections during an adaptive streaming session. As the client monitors its buffer, it may choose to upshift to a higher-bitrate profile or downshift to a lower one depending on how much the rate of video transport does or does not exceed the rate of media playout. Some adaptive streaming products open a new connection when upshifting or downshifting to a new profile.

Alternative Methods for Bitrate Adaptation

We have mentioned adaptive streaming methods that work based on the principle of switching between alternate encodings of a stream as a whole or switching between individual fragments of it encoded at several bitrates. There is also an alternative technology called Scalable Video Coding (SVC). In SVC-based systems, a client can choose media streams appropriate for the underlying network conditions and its decoding capabilities. SVC has been standardized as an extension to H.264/MPEG4 AVC video compression specification. SVC in its current form applies to video streams only.

In SVC, a video bitstream is made up of a hierarchical structure of layers. The base layer provides the lowest level of quality in terms of frame rate, resolution and signal-to-noise ratio (SNR). Each enhancement layer on top of the base layer provides an improvement for one or more of these scalable quality parameters. Enhancement layers can be independently stored or sent over the

network. Therefore, the overall stream bitrate can be modified by selectively adding or subtracting enhancement layers to/from a stream.

The quality knobs in SVC are referred to as temporal scalability, spatial scalability and SNR scalability. Temporal scalability provides a way to add or drop complete pictures to/from a stream. Temporal scalability is not something newly introduced by SVC. Temporal scalability in its most basic form has been used in traditional push-based streaming servers in terms of a method known as *stream thinning*. In spatial scalability, on the other hand, video signal is encoded at multiple resolutions. The reconstructed lower-resolution frames can be used to predict and reconstruct higher-resolution frames with added information sent in the enhancement layers. Finally, in SNR scalability, video signal is encoded at a single resolution, but at different quality levels by modifying the precision of encoding. Each enhancement layer increases the precision of the lower layers.

A key advantage of SVC lies in its ability to distribute information among various layers with minimal added redundancy. In other words, while a stream that is traditionally encoded at different quality levels has significant redundancy between the encodings; each layer in an SVC-encoded stream has minimal common information between the layers. This makes SVC efficient for storage of media at various quality levels. Another advantage of SVC is the graceful degradation of stream quality without client or server intervention when packets from enhancement layers cannot be delivered due to abrupt changes in network conditions. This is in contrast to multi-bitrate encoding where it requires switching from one encoding to another via a decision by the client or the server to adjust to the changes in network conditions. SVC streams, on the other hand, are typically more complex to be generated and impose codec restrictions compared to multi-bitrate streams. Thus, the adoption rate for SVC has been rather low.

Comparison of Push-Based vs. Pull-Based Streaming Protocols

Above, we described push and pull-based streaming protocols, mentioned their bitrate adaptation mechanisms and provided case studies. In this section, we present a qualitative comparison between these two classes of media streaming protocols.

Table 1 summarizes the differences between push and pull-based streaming[7]. One of the main differences between push and pull-based streaming is the complexity of the server architecture. As mentioned before, in pull-based streaming, the bitrate management is usually a task of the client, which significantly simplifies the server implementation. Furthermore, pull-based streaming can run on top of simple HTTP. Therefore, with minor provisions, an ordinary Web server can serve media content in pull-based streaming, although complexities may arise in streaming live content to

[7] In this comparison, pull-based streaming exclusively refers to streaming over HTTP. Albeit unlikely, a pull-based streaming method running over a different transport can be later developed and adopted.

a large number of clients. Push-based streaming, on the other hand, requires a specialized server that implements RTSP or a similar purpose-built protocol and has built-in algorithms for tasks such as bitrate management, retransmission and content caching. This makes pull-based streaming cost effective compared to push-based streaming.

Despite the lower server costs, pull-based streaming is usually less efficient overhead-wise than push-based streaming due to the underlying transport protocol being used. Compared to HTTP over TCP, RTP imposes a lower transmission overhead. Moreover, since RTP usually runs on top of UDP, the retransmission dynamics and congestion control mechanisms of TCP do not inherently exist in RTP.

Both protocols allow client buffering both at the beginning of a session and also after trick-play transitions such as fast forward to play. This is performed to prevent buffer underflows and achieve smooth playback experience. In adaptive streaming methods, the initial client buffering duration can be substantially less than non-adaptive streaming methods due to the fact the client can pick a lower-bitrate stream to begin with. This allows fast startup times and increases responsiveness.

One of the key benefits of push-based streaming is multicast support. Multicast allows a server to send a packet only once to a group of clients waiting to receive that packet. The packet is duplicated along the network path in the optimal way. A client can join or leave a multicast group on demand. This way, a client can receive only the packets it wants. Pull-based streaming, on the other hand, works based on the unicast delivery scheme. In unicast delivery, there is a one-to-one path between the server and each client. Therefore, in the worst case, a server has to send a packet as many times as the number of clients requesting that packet, and, similarly, a network node in the middle has to pass along the same packet multiple times. This reduces the efficiency of the network.

The routing and packet delivery over the network can be made more efficient by using content caching. The content is cached along the network on dedicated cache servers. This way a client may obtain content from a nearby cache server in network topology instead of going all the way up to the origin server. Probably, the most efficient use of caching is in the case of pull-based adaptive streaming. In this case, each fragment can be cached in the network independently.

Table 1: Comparison between push-based and pull-based streaming protocols.

	Push-Based Streaming	Pull-Based Streaming
Server Type	Streaming server Windows Media, Adobe Flash, Apple QuickTime, Cisco CDS	Web server LAMP, Microsoft IIS, Cisco CDS
URL Format	rtsp://..., rmtp://...	http://...
Protocols	RTSP, RTMP, RTP, UDP	HTTP

Bandwidth Usage	(Likely) More efficient	(Likely) Less efficient
Client Buffering	Yes	Yes
Video Monitoring and User Tracking	Standard tools	(Currently) Proprietary
Multicast Support	Yes	(Currently) No
Content caching	Requires special servers	Yes

Concluding Remarks

Popularity of watching traditional broadcast TV programming is weakening every day against the rapidly increasing interest in watching the content on the Web. Consumers are accessing the Web content not just from a TV in the living room but from a variety of devices from a variety of places connected through different types of access networks. Adaptive streaming methods that can deal with the challenges presented by the variety of such devices and networks as well as the scalability of content distribution to large audiences are further accelerating this trend transition, which will clearly impact the existing business models and create new revenue opportunities. In the second part of this article, we will look into applications for streaming including end-to-end mobile and in-home streaming, contrasting adaptive approaches to other video delivery paradigms, discuss the current standardization efforts and highlight the areas that still require further research and investigation.

References

1. *IPTV: Reinventing Television in the Internet Age.* **Thompson, Greg and Chen, Yih-Farn Robin.** May 2009, IEEE Internet Computing, Vol. 13, pp. 11-14 .

2. Transparent End-to-end Packet-switched Streaming Service (PSS); Protocols and Codecs. *3GPP TS 26.234.* [Online] Oct. 2010. http://ftp.3gpp.org/specs/html-info/26234.htm.

3. *Adaptive Streaming within the 3GPP Packet-Switched Streaming Service.* **Fröjdh, Per , et al., et al.** Mar. 2006, IEEE Network, Vol. 20, pp. 34-40.

4. Protected Interoperable File Format. [Online] May 2010. http://go.microsoft.com/?linkid=9682897.

5. ISO/IEC 14496-12 Base Media File Format. [Online] 2008. http://www.iso.org/iso/catalogue_detail.htm?csnumber=51533.

6. HTTP Live Streaming. *IETF Internet Draft.* [Online] June 2010. http://tools.ietf.org/html/draft-pantos-http-live-streaming.

Biographies

Ali C. Begen (abegen@cisco.com) is with the Video and Content Platforms Research and Advanced Development Group at Cisco. His interests include networked entertainment, Internet multimedia, transport protocols, and content distribution. He is currently working on architectures for next-generation video transport and distribution over IP networks, and he is an active contributor in the IETF in these areas. He holds a Ph.D. degree in electrical and computer engineering from Georgia Tech. He received the Best Student-Paper Award at IEEE ICIP 2003, and the Most Cited Paper Award from Elsevier Signal Processing: Image Communication in 2008. He is also a member of the ACM.

Tankut Akgul (akgult@cisco.com) is a software engineer in the Service Provider Video Technology Group at Cisco. His research interests are embedded software design, video compression and multimedia streaming. He currently designs and develops software for the next-generation IPTV set-top boxes. Prior to Cisco, he worked as a senior scientist at Soft Networks, LLC and developed several video codecs for Intel embedded SoCs. He received his Ph.D. degree in electrical and computer engineering from Georgia Tech. His Ph.D. thesis was on debugging and reverse execution of programs for embedded systems. He has several conference and journal publications in the area of embedded software.

Mark Baugher (mbaugher@cisco.com) is a technical leader in the Research and Advanced Development group at Cisco. Prior to Cisco, Mark co-founded and served as the CTO of Passedge Inc., which developed a multicast media security product. Over the years, Mark has led teams that delivered innovative technologies such as Intel's PC-RSVP, the first network reservation system on a PC, and LAN Server Ultimedia, IBM's first media server product. Mark has co-authored several widely used international standards in the IETF and ISMA, and he is currently co-chairing the Internet Gateway Device Working Committee in the UPnP Forum. Mark holds an M.A. degree in computer science from the University of Texas at Austin.

Watching Video over the Web, Part II

Applications, Standardization and Open Issues

Ali C. Begen, Tankut Akgul and Mark Baugher, Cisco

In this second part of a two-part article, the authors look into applications for streaming including end-to-end mobile and in-home streaming, contrasting adaptive approaches to other video delivery paradigms, discuss the current standardization efforts and highlight the areas that still require further research and investigation.

Applications: End-to-End Mobile and Home Streaming

There are many video applications and services on the Internet, and the first part of this article mentioned several of them. The most important of these applications arguably fall into the mobile and home entertainment categories. These two categories span the range of video presentation formats for portable, standard and high-definition video. They also operate on diverse networks and use both managed and unmanaged end-to-end network services. Mobile and home video applications are among the most popular uses of video and include both walled-garden and Internet applications. This section identifies mobile and home-streaming use cases and considers the suitability of pull-based versus push-based adaptive streaming for each use case.

Mobile Streaming

People stream movies, YouTube videos and other media on the road, in hotels, on campuses and while in motion to their mobile devices. Portable-definition (PD) encoding is a video format for mobile devices, which is a lower resolution and lower-bitrate encoding than standard (SD) and high-definition (HD) encoding, which are common in homes. PD video delivery scales down to the lower speeds and higher-loss found on most mobile networks: Variations in load on network cells, changes in radio-network conditions and fluctuations in available throughput force video applications to adapt to the network rather than *vice versa* in GPRS, EDGE, UMTS and CDMA 1x networks. Thus, it is no surprise that mobile handset vendors and operators have been in the forefront in defining and developing adaptive streaming standards.

When the mobile connection is to a walled-garden server, the network provider can manage network video delivery to a service-compliant, walled-garden client. This is one use case. If the end-to-end connection is to a server on the Internet, the network provider cannot provide QoS

management, and the service is not managed. This is another use case. There is also a third use case that combines mobile and home streaming.

Walled Garden

In this case, the end-to-end path is contained in the cellular provider's network. The network operator can specialize service for video delivery. The provider can choose to use push-based adaptive streaming services from specialized streamers to get optimal use of the network resources and improved video quality over a range of network conditions. Alternatively, the provider can choose to reuse their data network services such as HTTP servers and caches, and employ pull-based adaptive streaming methods such as Apple's iPhone HTTP Live Streaming.

Mobile Internet

The cellular provider cannot manage the service quality on an end-to-end basis when the video connection traverses the public Internet. In this case, pull-based adaptive streaming has advantages over push-based methods.

Hybrid Mobile-Home Use Case

The cellular provider cannot guarantee a service when the first network hop(s) or the last network hop(s) are on the client's home network. Femtocell networks and home-network services to mobile clients are examples. When an unmanaged home network is a hop on the end-to-end path, and particularly when it is a wireless hop, pull-based adaptive streaming is desired instead of push-based methods.

Home-Network Streaming

The home network use cases are crucial to video streaming since video is so widely used by consumers at home. Although in many residences, the home network consists of a single Ethernet cable connected to a broadband modem and the home PC, more complex home networks are becoming common today in many countries. Increasingly, the self-installed home LANs are capable of transporting video.

802.11 (WiFi) LANs are widespread in many regions of the world. One well-known video-delivery problem in WiFi networks is the divergence of network speed and reliability due to network topology, load, and other variables. The presence of metal objects, common home appliances and other sources of interference mean that a television in one room often cannot sustain the same quality of presentation as a device in another. The reality of diverse network capacity among receivers coupled with the design of the 802.11 medium access control also cause a problem for multicast streaming over wireless LANs (WLAN) (1).

Video-on-demand, not broadcast TV, is likely to be the dominant video application on the home networks. Unfortunately, wireless and other types of common home LANs such as powerline, are

vulnerable to interference and overload. Although the migration of WiFi products to 5 GHz from 2.4 GHz reduces some of the interference problems, even 802.11n is vulnerable to interference that results in widely differing loss and throughput characteristics on the WLAN (2). Another home LAN technology that is vulnerable to interference is powerline communications (PLC) in the home. In fact, with the exceptions of well-laid Ethernet over Cat5 or over cable (as defined by the Multimedia over Coax Alliance - MoCA), it is hard to imagine a home networking technology that can offer uniform, high-speed services throughout the home.

Whereas HTTP adaptive streaming is productively used on the public Internet and unmanaged networks in general, adaptive streaming is arguably crucial for unicast, point-to-point video delivery on home WLANs. Home networks are mostly unmanaged and exhibit great differences in network speeds and feeds available to a diverse set of devices. It is unreasonable to expect that a typical home user will manage and select particular resolutions of a movie title based upon the quality of network service that a particular device can obtain from a particular spot on the home network.

It is therefore unlikely that managed video will ever be common for in-home streaming. Managed video will likely continue to provide a superior, trouble-free and gold-level video service to the home, however, which unmanaged services try to replicate. In the use cases that follow, both managed and unmanaged methods are considered, and push-based streaming is compared to its pull-based counterpart.

Streaming to a Home Client from a Home Server

This use case is arguably the most interesting but the least common case: For years, hobbyists and digital media enthusiasts have connected their home TVs to their home networks to play movies from a PC, gaming console or other servers, but this is not yet a mainstream practice. In the U.S., the WiFi network can automatically connect an off-the-shelf home network attached server to a Sony PlayStation and a host of other devices implementing Digital Living Network Alliance (DLNA) specifications[1]. However, this capability is not widely used by non-technical consumers. Setting up a home media server requires sufficient knowledge of how to load a server with commercial movies, which are usually encrypted, match formats with players, manage Digital Rights Management (DRM) when it is exposed or not circumvented, and perform other tasks that people outside of the computer industry do not routinely do. Nonetheless, home servers can be quite useful to people who want to play their digital movies without experiencing the bottlenecks of the Internet or service provider's network. The home user should be able to play a file even when their broadband network connection is down, and play their movies on all the TVs or other displays in

[1] http://www.dlna.org

the home besides just one. iTunes has achieved in-home music sharing; Apple and other vendors will undoubtedly develop video sharing among home devices as well. On the other hand, standard solutions such as DLNA can open the market to a greater diversity of products and are likely important to future in-home video use case. Streaming to a home client from a home server is a use case that is arguably more important to the future of home video than it is today.

Whether using proprietary or standard players, home wireless networks are limited by the available bandwidth of the radio spectrum and various types of noise and interference; wireless networks, in general, are arguably the weakest link in the video transmission chain. The quality of the video presentation on a wireless home network is subject to all of the problems of an unmanaged network plus problems with signal loss, interference, and contention with other traffic on the WLAN (3).

When push-based adaptive streaming is used, the home-network owner must install, configure and manage a specialized push-based streaming server on the home network that is not managed by a business such as the Internet service provider. Moreover, the most popular protocol for video on the home network is HTTP followed by RTP/RTSP. HTTP is used on the Web and in home networking protocols such as DLNA. It will be used by many new services such as Ultraviolet. There are many *turnkey* HTTP video applications and commodity servers available in the market, and HTTP works through home firewalls. Thus, pull-based streaming over HTTP has advantages over push-based streaming over RTP.

Streaming to a Home Client from an Internet Server

This use case includes, Webcasts, YouTube, Hulu, iTunes and other Internet video services to a home PC or digital TV. As explained above, a potentially troublesome home network hop, such as a home WLAN, will often be part of the end-to-end video path whenever the network supports more than a single device connected to a modem; a configuration that is becoming increasingly rare in developed countries. Managed video within the home is unlikely to become common since the home network is usually unmanaged. The public Internet, moreover, only supports unmanaged video delivery. Thus, pull-based adaptive streaming is the only feasible approach for this use case.

Streaming to a Home Client from a Managed Server

Most digital television services today, including video-on-demand, originates from video service providers. A provider's video service might be an ATSC, IPTV or HTTP service, which in all cases can be optimized for managing video over the provider's network to a home device. Managed video generally achieves a level of quality that unmanaged video attempts to meet – often without success. Thus, a managed server can use a push-based adaptive streaming method.

HTTP progressive download or adaptive streaming can of course run over a managed video network and even use a managed server. Thus, this use case includes both push-based and pull-based streaming.

Streaming to a Home Client via Peer-to-Peer (P2P) Delivery

In certain parts of the world, particularly Asia and some parts of Europe, a popular method for streaming video is to use P2P networks. Research has shown that P2P transport could reduce the load on the source servers and provide a better scalability for large streaming populations. However, with the advances in server and caching infrastructure designs, large streaming capacities are now easily achievable without needing P2P transport. On the business side, uncontrolled and unmanaged P2P systems often fail to provide a fair revenue sharing between the content and Internet service providers. While there are industry-wide efforts to address some of the specific issues related to network and provider-friendly P2P transport, the business models that will be profitable for both the content owners/providers and Internet service providers are still not in place. Thus, in this article, we will not go into further details; interested readers may refer to the December 2007 issue of the IEEE Journal on Selected Areas in Communications for a detailed coverage of various topics in P2P streaming.

Summary

The two major trends in Internet video, therefore, are streaming to mobile devices and to a TV or other in-home screen. Of the six use cases of mobile and home-network streaming, most have unmanaged networks along their end-to-end paths.

Two of the cases shown in Table 1 can benefit from push-based adaptive streaming using specialized servers and clients, i.e., the (walled-garden) managed server use cases in which the end-to-end path is over a provider-managed network. Managed networks are increasingly optimized for HD video delivery; managed network devices comply with the technical protection measures required by major video producers. And video service providers have the strategic business relationships to allow them an early spot in the windowed movie release schedules as well as special pay-per-view video events. This gives service providers a role for premium content delivery of major titles, particularly expensive movies. As stated above, managed video is the standard that Internet video must meet.

Table 1: End-to-end paths and preferred streaming methods.

	Internet Server	Managed Server	Home Server
Mobile Client	Pull-based streaming	Push and pull-based streaming	Pull-based streaming
Home Client	Pull-based streaming	Push and pull-based streaming	Pull-based streaming

How close can unmanaged video come to managed-video quality? And how well can such services scale? These questions are crucial to the future of Internet video services that use pull-based adaptive streaming. Arguably, the Internet server path in Table 1 offers users the greatest access to the most content – including broadcasters' sites, iTunes, Netflix, Hulu, and even social networking sites. If pull-based adaptive streaming can match the quality of managed video well enough, then this use will favor the future success of this video streaming technology.

Of the six use cases, streaming from a home server to a home client is much less popular compared to Netflix, Hulu, iTunes, and managed video from service providers/retailers. This use case becomes more important, however, as digital files replace digital disks in the home. And when the broadband connection becomes unavailable, as broadband connections sometimes do, then the user should at least have access to their titles within the bounds of a functioning home network. Streaming to a home client is also interesting because this use case operates on the home network, and the video-delivery performance characteristics of home networks such as 802.11 are important to the Internet and managed server cases as well. We therefore conclude that pull-based adaptive streaming methods have considerable advantages over push-based adaptive methods for most Internet video applications.

Standardization Efforts

Microsoft PIFF (Protected Interoperable File Format) (4) and SST (Smooth Streaming Transport) (5) are complete specifications for an adaptive-streaming container and streaming protocol. PIFF and SST use the standard MP4 fragmented file structure. Microsoft has also opened up these specifications for widespread use by offering the Microsoft Community Promise, which provides intellectual property rights guarantees for organizations that choose to use their adaptive streaming protocols. Apple has similarly offered its HTTP Live Streaming on the Web and in an IETF draft (6). Both the Microsoft and Apple specifications reflect the current adaptive streaming products that are in the market and offered by the two companies.

These and other specification efforts are appearing in various venues. These include 3GPP, MPEG and industry consortia such as the Digital Entertainment Content Ecosystem (DECE) LLC, which

announced a new service called *Ultraviolet* in July 2010. DECE has not made public its technical specification at the time of the writing of this article. It has been announced, however, that the DECE Ultraviolet service would support a common container format, movie downloads and streaming with support for multiple DRMs[2]. It seems likely that the final specifications will support at least some of the state-of-the-art media distribution technologies that have been considered in this article.

Although DECE LLC's membership includes dozens of companies who are involved in media distribution, it is a private effort by a consortium of companies. Adaptive streaming technologies have made their way into public standards development organizations including 3GPP and MPEG.

3GPP is the first organization that released a specification related to adaptive streaming over HTTP. 3GPP Release 9 (7) was published in March 2010, and the SA4 Working Group is currently in the process of bug-fixing this specification taking into account experience from the first implementations. In addition, 3GPP is preparing an extended version for Release 10, which is scheduled to be published later in 2011. This release will include a number of clarifications, offer improvements and add new features. On the MPEG side, the Streaming of MPEG Media over HTTP Ad Hoc Group has received 15 submissions to their call for proposals published in May 2010. After an initial evaluation period in July 2010, this ad hoc group, which has been renamed as Dynamic Adaptive Streaming over HTTP (DASH) Ad Hoc Group, has adopted the 3GPP Release 9 as a baseline specification and started running several evaluation experiments. The DASH Ad Hoc Group will work on standardizing the manifest file, delivery format, conversion from/to existing file formats and the use of MPEG2 Transport Streams as a media format. The DASH Ad Hoc Group will also coordinate closely with 3GPP SA4 Working Group to better align the respective specifications.

Open Issues and Future Directions

At the writing of this article, there are still many questions about adaptive streaming that have not been adequately answered yet. While there are early investigations such as (8) and the references therein, most of the questions still remain open mostly due to the lack of field data required to conduct a rigid analysis. To date, TCP and HTTP have been individually studied in great detail for conventional applications. Proposals have been made for both protocols to make them a better fit for streaming-like applications supported by simulations and experiments. However, making generalizations and drawing conclusions for large-scale streaming deployments based on narrowly scoped studies is not trivial and may be easily misleading.

[2] http://www.homemediamagazine.com/digital-copy/dece-becomes-ultraviolet-20060

Historically, TCP has been designed and optimized for FTP-like applications, and implications of its congestion control algorithm and currently adopted best practices such as Nagle's algorithm, delayed acknowledgments, slow-start restart (See RFC 5634) have to be examined in the context of adaptive streaming. Effects of running multiple TCP connections simultaneously for overcoming the head-of-line blocking problem or another transport protocol that inherently provides a better support for multiple streams such as SCTP have also to be studied carefully. Most importantly, these studies must represent large-scale deployments with many origin servers, caches and client streaming a variety of content over diversely characterized network paths.

A niche area is to develop instrumentation tools to assess the effectiveness and performance of adaptive media transport. These tools must be user-friendly, able to provide adequate information for diagnostics and fault isolation. Such information becomes quite handy for service providers in fixing problems and making the necessary provisions and enhancements in their networks.

An interesting future research direction is to understand how network elements can help a provider achieve a better performance, which could be supporting more clients or delivering their clients a better and more consistent streaming quality. In other words, rather than considering the underlying network as a black box or just bunch of pipes that carry the bits, having the endpoints (e.g., origin servers and clients) and middle boxes (e.g., cache servers) talk to the network, exchange information and act accordingly may potentially provide many benefits. For example, the network itself is first to know about a congestion or failure. Such information certainly helps the servers and clients adapt faster and more accurately.

Last but not least important open issue is whether the network capacity in a multi-access network can sufficiently support many clients concurrently. In such a scenario, each client competes with others for more bandwidth and is likely experience a continual bitrate fluctuation, which adversely impacts quality of experience. Prepositioning the content or using multicast (especially for live streaming) may prove to be useful.

In a future article, we hope to provide the results from our ongoing studies investigating these open issues.

Acknowledgments

We thank our colleagues at Cisco for their reviews and feedback for this article.

References

1. *Multicast in 802.11 WLANs: an experimental study.* **Dujovne, Diego and Turletti, Thierry .** Torremolinos, Malaga, Spain : ACM, 2006. ACM MSWiM.

2. *Understanding and Mitigating the Impact of RF Interference on 802.11 Networks.* **Gummadi, Ramakrishna , et al., et al.** Kyoto, Japan : ACM, 2007. ACM SIGCOMM.

3. *Performance Evaluation of Video Streaming with Background Traffic over IEEE 802.11 WLAN Networks.* **Cranley, Nicola and Davis, Mark .** Montreal, QC, Canada : ACM, 2005. ACM WMuNeP.

4. Protected Interoperable File Format. [Online] May 2010. http://go.microsoft.com/?linkid=9682897.

5. Smooth Streaming Transport Protocol. [Online] Sept. 2009. http://go.microsoft.com/?linkid=9682896.

6. HTTP Live Streaming. *IETF Internet Draft.* [Online] June 2010. http://tools.ietf.org/html/draft-pantos-http-live-streaming.

7. Transparent End-to-end Packet-switched Streaming Service (PSS); Protocols and Codecs. *3GPP TS 26.234.* [Online] Oct. 2010. http://ftp.3gpp.org/specs/html-info/26234.htm.

8. *An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP.* **Akhshabi, Saamer, Begen, Ali C. and Dovrolis, Constantine.** San Jose, CA : ACM, 2011. ACM MMSys.

Biographies

Ali C. Begen (abegen@cisco.com) is with the Video and Content Platforms Research and Advanced Development Group at Cisco. His interests include networked entertainment, Internet multimedia, transport protocols, and content distribution. He is currently working on architectures for next-generation video transport and distribution over IP networks, and he is an active contributor in the IETF in these areas. He holds a Ph.D. degree in electrical and computer engineering from Georgia Tech. He received the Best Student-Paper Award at IEEE ICIP 2003, and the Most Cited Paper Award from Elsevier Signal Processing: Image Communication in 2008. He is also a member of the ACM.

Tankut Akgul (akgult@cisco.com) is a software engineer in the Service Provider Video Technology Group at Cisco. His research interests are embedded software design, video compression and multimedia streaming. He currently designs and develops software for the next-generation IPTV

set-top boxes. Prior to Cisco, he worked as a senior scientist at Soft Networks, LLC and developed several video codecs for Intel embedded SoCs. He received his Ph.D. degree in electrical and computer engineering from Georgia Tech. His Ph.D. thesis was on debugging and reverse execution of programs for embedded systems. He has several conference and journal publications in the area of embedded software.

Mark Baugher (mbaugher@cisco.com) is a technical leader in the Research and Advanced Development group at Cisco. Prior to Cisco, Mark co-founded and served as the CTO of Passedge Inc., which developed a multicast media security product. Over the years, Mark has led teams that delivered innovative technologies such as Intel's PC-RSVP, the first network reservation system on a PC, and LAN Server Ultimedia, IBM's first media server product. Mark has co-authored several widely used international standards in the IETF and ISMA, and he is currently co-chairing the Internet Gateway Device Working Committee in the UPnP Forum. Mark holds an M.A. degree in computer science from the University of Texas at Austin.