Adaptive Streaming — a brief tutorial

Niels Laukens
VRT Medialab

The Internet and worldwide web are continuously in motion. In the early days, pages were pure text although still images were incorporated fairly quickly, followed by moving pictures in the form of "Animated GIF" files. True video only became possible years later.

Nowadays, video playback is ubiquitous on the web, but a smooth playback experience is not always guaranteed: long start-up times, inability to seek to an arbitrary point and re-buffering interruptions are no exceptions. In the last few years, however, new delivery techniques have been developed to resolve these issues, in particular "Adaptive Streaming" as described in this article.

To communicate over the Internet, one uses the Internet Protocol (IP), usually in association with the Transmission Control Protocol (TCP). IP is responsible for getting the packet across the network to the required host. TCP guarantees that all packets arrive undamaged in the correct order – if necessary, reordering out-of-order packets and/or requesting a retransmit of lost packets. This is essential for transmitting files reliably across the Internet, but these error-correcting powers come at a cost. TCP will refuse to release anything but the next packet, even if others are just waiting in its buffer (this can happen for example when TCP is waiting for a retransmitted packet to arrive). While this is the desired behaviour when downloading a document, the playback of video and audio are special in this sense: it is usually preferred to skip the missing packet and introduce a short audible and/or visible glitch, as opposed to stalling the playback until the missing packet arrives.

The Real-time Transport Protocol (RTP) chooses not to use TCP's error correction: it uses the User Datagram Protocol (UDP) instead. UDP only filters out damaged packets, but does not attempt any error correction at all. RTP is still one of the most popular formats for streaming audio and video, especially in IP telephony (Voice over IP, VoIP) and the video contribution world. Due to its lack of error resilience, RTP is mostly used on managed, internal networks. In addition, most firewalls connecting users to the Internet are configured not to allow UDP, and hence RTP traffic.

The Hyper Text Transfer Protocol (HTTP) is used to serve pretty much every website on the Internet. HTTP is allowed by the majority of firewalls, although sometimes a proxy server is enforced. Therefore, this is a very attractive protocol to deliver items, including video, to a large audience. The simplest way to deliver video over HTTP is called ***HTTP downloading***. The multimedia file is simply handled as any regular file and transmitted across the Internet with the error-correcting magic of TCP. Once the file has completely downloaded, it can be played – usually in a stand-alone player, although a browser plug-in is also possible. While this technique guarantees a completely seamless playback at optimal quality, the user needs to wait until the file has completely downloaded before it can be played.

A better user experience is achieved by preparing the multimedia file upfront. These preparations move all information required to start the playback to the beginning of the file. That way, a smart client can start playback, even while the file is still downloading! There are no special requirements on the server side: it is still served as a regular file.

This technique is called ***progressive downloading***. Obviously, the download should be fast enough to stay ahead of the playback; otherwise the dreaded rebuffering will happen. In this scenario, the user's ability to seek to an arbitrary time in the playback is reduced: it is only possible to seek to the part that has already downloaded.

Of course, this limitation was quickly overcome by a slightly adapted variant, using a smarter server that understands how the multimedia file is structured internally. In ***HTTP pseudo-streaming***, clients can request a specific time interval to download. Seeking to your favourite scene thus becomes possible without downloading the full movie. However, once the client starts downloading, it still downloads as fast as possible (as is the case with all HTTP downloads), wasting a lot of bandwidth when the video is not watched fully. But even if the client or server would limit the download speed to be "just fast enough", there are still issues.

All of the above techniques use a single "conversation" to transfer all required media data. This causes problems when moving around: when a smartphone moves out of range of a Wi-Fi hotspot, it can automatically switch over to a cellular data connection (UMTS, EDGE, GPRS …). This however causes all active conversations to be interrupted. They need to be restarted on this new connection[1]. On top of that, this new cellular connection is very likely to provide less bandwidth, possibly too little to continue playback of the high-bitrate video you started on Wi-Fi.

Streaming or downloading

There is a lot of debate over whether these techniques should be called "streaming" or "downloading". Especially in the view of rights management, this is a very important distinction: usually broadcasters are allowed to stream the content, but are not allowed to make the content available for downloading.

For the most part, this is not a technical issue but a matter of definition:

- *It is streaming if only the parts of the media that are actually needed are transferred*: in this sense, the adaptive streaming techniques are rightfully named "***streaming***".
- *As soon as a media file is copied to the client, it's downloading*: Since the HTTP protocol deals with files, the techniques described should be called "***adaptive downloading***".
- *Streaming makes it impossible for a user to store a copy of the media*. This definition is useless: it's always possible to store a copy: just aim a camcorder at your screen! It can be a useful definition if it specifies how hard it must be to store a copy.
- *The media is never stored (cached) on the client*. This definition also needs clarification: how long (or short) can it be stored? Since most encoding algorithms reuse pieces of previous frames, they need to store that frame for a limited time. Surely streaming should not be prevented from using Long-GoP formats!

From a technical point of view, "streaming" is a defendable terminology, but the legal department might disagree.

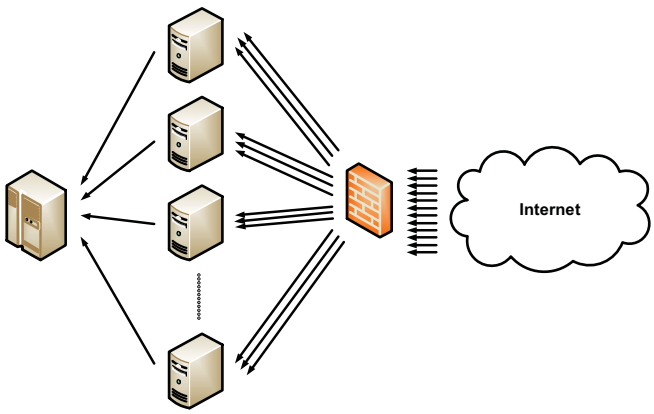


Figure 1
Requests are spread over the available edge servers. The initial request will be forwarded to the origin server. Once the response is retrieved, it is stored in a caching server at the edge and passed on to the requesting client. All subsequent requests for this object will be served from the cache. The object is removed from the cache when it's no longer valid (as specified by the origin server). Valid objects may also be deleted by the cache management to free up space for more popular objects.

1. IP mobility allows conversations to be moved instead of restarted, but is hardly used in practice.

This is where adaptive streaming steps in. It combines a lot of the qualities of the aforementioned protocols, while avoiding their pitfalls. Although adaptive streaming techniques can run over a wide variety of protocols, they typically run over HTTP, which gets them through most firewalls without much hassle. Using HTTP has even more benefits: caching functionality is built in to the core of the protocol. Every HTTP object contains a tag that specifies how long this object is valid: if it is requested again during this time span, the saved copy may be used without contacting the original server. This can relieve the origin servers by spreading the load to the edges (proxy servers or specialized caching servers), as illustrated in *Fig. 1*.

On the server-side, using HTTP has an additional benefit: it's a stateless protocol. In a stateless protocol, the server doesn't store any persistent information about the client or its requests. In other words, once a request is handled, the server resets itself to its original state[2]. This is very convenient for load balancing: every request can be handled by any available server, without keeping track of which server "owns" that particular client.

Adaptive streaming

From a server perspective, the basic principle behind adaptive streaming techniques is fairly simple: provide the clients with a table of URLs. Every URL points to a specific time interval (the columns) of a specific quality (the rows) of the same content, as illustrated in *Fig. 2*. All intelligence is implemented in the client; the server can be any HTTP-compliant device serving regular files.

Figure 2
The media file is created in multiple qualities, here represented as rows. It is also cut into time intervals (here represented by the columns) synchronously across the different qualities. Every individual chunk is individually addressable by the client. This example uses fixed-size chunks, but that is not required.

Once the client has downloaded the table of URLs, it needs to employ its knowledge of the client system to select the appropriate URL to fetch next. Obviously, the client should start playback of the first time interval, but which quality should be used? Usually, a client will start with the lowest available quality. This will give the fastest start-up time. During this first download, the available network bandwidth can be estimated: e.g. the first chunk was 325 kB large and was downloaded in 1.3 seconds; the network bandwidth is thus estimated to be 2 Mbit/s. Clients then usually switch up to the highest available quality within their network bandwidth. Seeking to an arbitrary point is also possible: the client can calculate the corresponding time interval and request that segment right away.

2. If there is a need for persistent storage, it needs to be handled on top of HTTP. Servers typically use a database to store persistent information.

Implementations can use more metrics than just the bandwidth. A client could detect the current playout resolution. That way a thumbnail player inside a web page could download the low-resolution version, even though the network bandwidth would allow a better quality, only switching to the HD variant once the video is played at full screen. Clients could also take the available CPU power (or hardware support) into account, avoiding stuttering playback even if the bandwidth would allow a higher bitrate.

To achieve this seamless playback and switching, there are obviously some requirements on the individual chunks. Since clients can zap in to an arbitrary chunk at an arbitrary quality, each and every chunk must be completely self-contained. Modern video encoding algorithms use “inter-frame compression”: they reuse pieces of previous frames to construct the current frame, only transmitting the differences. Obviously, the receiver needs to have access to these previous frames. Usually, an “Intra frame” (I-frame) is inserted every now and then. I-frames do not reference any previous frame. Hence, every chunk must start with an I-frame (i.e. they must start on a GoP boundary). And since the following chunk needs to start with an I-frame as well, a chunk will always end one frame before an I-frame (i.e. at another GoP boundary). The GoP length thus needs to be balanced: longer GoPs allow higher compression, but slower switching.

The same story is true for the audio. Audio is usually encoded by transforming a sequence of samples (e.g. AAC uses 2048 samples per block). The chunk cuts can only happen on an integer multiple of this block size. Things get even more complicated when different qualities have different frame rates (or sample rates in the case of audio). Care must be taken that the chunk boundaries align properly.

It is worth mentioning that the adaptive streaming protocols only define the wire format (i.e. the bits and bytes that are transmitted over the network): they don't imply anything on the server side. They require that every chunk must be *addressable* individually. This can be accomplished by having individual files for every chunk, but this is not required. In fact, Microsoft’s implementation uses a single (prepared) file for the full duration. A server module picks out the correct byte ranges to serve individual chunks to clients. The protocols require every chunk to be individually decodable, hence excluding SVC. But servers may use SVC internally to conserve disk capacity or backhaul bandwidth, and transcode when they output the chunks to the users.

SVC – Scalable Video Coding

In contrast to traditional encoding schemes, SVC produces a bitstream that contains one or more subset-bitstreams. These subsets can be decoded individually, and represent a lower quality (either in spatial resolution, in temporal resolution or in compression artefacts). SVC thus transmits a “base layer” and one or more “enhancement layers”. This conserves bandwidth and/or storage requirements, if all qualities are needed. For an individual quality however, SVC typically requires 20% more bits compared to a single-quality encoding, because of the intermediate steps.

Abbreviations

3GPP	3rd Generation Partnership Project	**IPTV**	Internet Protocol Televison
AAC	Advanced Audio Coding	**OIPF**	Open IPTV Forum
CPU	Central Processing Unit	**RTP**	Real-time Transport Protocol
EDGE	Enhanced Data rates for GSM Evolution	**SVC**	(MPEG-4) Scalable Video Coding
F4V	An open container format for delivering synchronized audio and video streams	**TCP**	Transmission Control Protocol
		UDP	User Datagram Protocol
FMP4	(FFmpeg) MEncoder MPEG-4 video codec	**UMTS**	Universal Mobile Telecommunication System
GoP	Group of Pictures		
GPRS	General Packet Radio Service	**URL**	Uniform Resource Locator
GSM	Global System for Mobile communications	**VoIP**	Voice-over-IP
HTTP	HyperText Transfer Protocol	**WMA**	(Microsoft) Windows Media Audio
IP	Internet Protocol	**XML**	eXtensible Markup Language

Current implementations

There are currently three major players pushing their implementation: Microsoft, Apple and Adobe. Microsoft has incorporated its "Smooth Streaming" into its Silverlight player. Apple has implemented "HTTP Adaptive Bitrate Streaming" in both its desktop products (since Mac OS X 10.6 Snow Leopard) and its mobile products (since iOS 3). Adobe uses its Flash player (v10.1 and up) to deliver "HTTP Dynamic Streaming". Obviously, the different vendors each have their own implementation but, surprisingly, they have enough in common to allow some clever reuse.

- **Microsoft's** implementation[3] uses an XML "Manifest" file to communicate the table of URLs to the client. The URLs are then formed by filling in a supplied template with the required parameters. Each chunk contains either audio or video material in a "fragmented MP4" container. Audio and video are thus requested separately and can be switched to different qualities separately. The "fragmented MP4" format is, as the name implies, a variant of the ISO specification. Although the current Silverlight player supports only H.264 and VC-1 video material, and AAC and WMA audio material, the spec. itself is codec-agnostic.

- The **Apple** variant[4] communicates the table of URLs by using a hierarchical text file called a "Playlist". The top-level playlist contains the list of available qualities, each with their individual sub-playlist. This sub-playlist explicitly enumerates the individual URLs for each chunk. These chunks contain both audio and video in an MPEG-2 Transport Stream format, a well-known format in the broadcast world. Here, the specification also allows for any video and audio codec, but current implementations are limited to H.264 video and AAC or MP3 audio.

- **Adobe** uses[5] yet another XML format. It has its own MP4 variant which also fragments the metadata. The Flash player only supports H.264 and VP6 video and AAC and MP3 audio. As is the case with the previous two, the spec. itself is codec-agnostic.

Besides these three "proprietary" implementations, several standards bodies are working on an "open" standard:

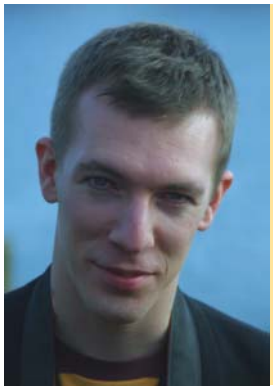
- The **3GPP** has joined forces with the **OIPF** and have defined[6] their "Dynamic Adaptive Streaming over HTTP". It's also based on an XML file to communicate the URLs, and is codec-agnostic. They reuse the already existing 3GPP container format. As far as I know, there are currently no implementations of this standard.
- **MPEG** is currently drafting "Dynamic Adaptive HTTP Streaming". It is considering both the 3GPP and the Microsoft proposal for standardization.

Towards unified streaming

Beside the obvious differences in the current implementations, there are some similarities as well: the three major players all support H.264 for the video codec; AAC is the common audio codec. We can use this fact to our advantage and encode the content only once (per quality). The raw video and audio streams are then wrapped in a container (e.g. fMP4, F4F or MPEG-2 TS). From a computational point of view, encoding is very intensive, while (re)wrapping is very simple.

We can take this idea one step further and choose a single disk format: the encoders output to this common format, which is re-wrapped into the correct format on-the-fly upon request. This effectively reduces the required disk space by a factor of three, at the cost of a slightly increased workload. This solution is actually used by Microsoft's IIS server: it can be configured to (also) stream in the

3. http://learn.iis.net/page.aspx/684/smooth-streaming-transport-protocol
4. http://tools.ietf.org/html/draft-pantos-http-live-streaming-04
5. http://help.adobe.com/en_US/HTTPStreaming/1.0/Using/WS9463dbe8dbe45c4c-1ae425bf126054c4d3f-7fff.html
6. http://www.3gpp.org/ftp/Specs/html-info/26247.htm



As a telecommunications engineer, **Niels Laukens** has an academic background in modern telecommunication systems, including IP-multicast, cryptographic and other technologies. His master thesis on unicast distribution control for multicast transmissions received awards on several occasions. During his first job as a networking and security expert, he obtained hands-on experience of the possibilities and limitations of real-life IP networks.

Currently, as a senior researcher in the R&D department of VRT, Mr Laukens works on broadband distribution issues. His main focus is on the back end of the distribution chain, encompassing encoding and server software, and on back-end architecture design and development. Recent projects include adaptive streaming technologies and scalable server architectures.

Apple format. Apple-compatible chunks are created automatically from their Smooth Streaming equivalents by the server.

We can go one step further, and also unify the streaming format. The rewrapping can be done by the client. In this scenario, it's most probable that the format will be Apple's, since there is no easy way to incorporate additional rewrapping logic in the iPhone/iPod/iPad product family. Both Silverlight and Flash have embedded support for application logic, so rewrapping code can be delivered along with the player. While this might seem far-fetched, a Dutch company Code Shop already provides a commercial module that plays a Smooth Stream inside a Flash player.

Conclusions

Adaptive streaming techniques are a major step towards a good quality of experience for video delivery over the public Internet. It uses the standard HTTP protocol to be firewall-compatible. In addition, HTTP provides extensive caching capabilities which allow the delivery to scale much easier than other protocols. All adaptivity is implemented on the client side, which can use whatever metrics available to it. All implementations use network bandwidth, but screen resolution, CPU power or user preferences are also possible.

Future evolution of these techniques may provide better support for variable-bitrate (VBR) streams. Currently the qualities are denoted by a single number representing the average bitrate. But when streams vary widely in bitrate to accommodate varying image complexity, things can go wrong: a client estimating 5 Mbit/s of network bandwidth may try to download the 4 Mbit/s-on-average version, only to find out that this particular time chunk is actually 9 Mbit/s, hence failing to download in time.

This version: 4 February 2011

Published by the European Broadcasting Union, Geneva, Switzerland

ISSN: 1609-1469

Editeur Responsable: Lieven Vermaele

Editor: Mike Meyer

E-mail: tech@ebu.ch