HTTP ADAPTIVE STREAMING WITH MEDIA FRAGMENT URIS

Wim Van Lancker, Davy Van Deursen, Erik Mannens, Rik Van de Walle

Ghent University – IBBT
Ghent, Belgium
{wim.vanlancker, davy.vandeursen, erik.mannens, rik.vandewalle}@ugent.be

ABSTRACT

HTTP adaptive streaming was introduced with the general idea that user agents interpret a manifest file (describing different representations and segments of the media); whereafter they retrieve the media content using sequential HTTP progressive download operations. MPEG started with the standardization of an HTTP streaming protocol, defining the structure and semantics of a manifest file and additional restrictions and extensions for container formats. At the same time, W3C is working on a specification for addressing media fragments on the Web using Uniform Resource Identifiers. The latter not only defines the URI syntax for media fragment identifiers but also the protocol for retrieving media fragments over HTTP. In this paper, we elaborate on the role of Media Fragment URIs within HTTP adaptive streaming scenarios. First, we elaborate on how different media representations can be addressed by means of Media Fragment URIs, by using track fragments. Additionally, we illustrate how HTTP adaptive streaming is realized relying on the Media Fragments URI retrieval protocol. To validate the presented ideas, we implemented Apple's HTTP Live streaming technique using Media Fragment URIs.

Index Terms— HTTP Streaming, Media Delivery, Media Fragment URIs

1. INTRODUCTION

Multimedia content has become an essential part of the World Wide Web. Moreover, Web-based media is exploding: it is used for entertainment, education, advertising, product reviews, etc. Media delivery on the Web evolved from download-and-play over progressive download to real-time streaming protocols such as the Real Time Streaming Protocol (RTSP). Recently, a new media delivery technique, called HTTP adaptive streaming, was introduced showing an interesting combination of the features of real-time streaming protocols and HTTP progressive download.

The research activities as described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), and the European Union.

Various proprietary implementations are already available: Microsoft's Smooth Streaming, Apple's HTTP Live streaming, and Adobe's Dynamic HTTP Streaming. Almost all current proprietary solutions for HTTP streaming define the structure and semantics of a manifest file, describing the high-level structure of the media content in terms of representations and temporal segments. Additionally, extensions and restrictions are defined for one or more existing container formats encapsulating the media content. User Agents (UA) interpret the manifest file and retrieve the media content using sequential HTTP progressive download operations. Currently, MPEG is standardizing HTTP adaptive streaming as media delivery protocol, Dynamic Adaptive Streaming over HTTP (DASH, [1]), which is based on 3GPP Adaptive HTTP Streaming.

In this paper, we investigate how Media Fragment URIs can be used within HTTP adaptive streaming scenarios. Note that Wu *et al.* already indicated the relevance of Media Fragments within HTTP streaming [2]. The specification of Media Fragment URIs is currently being developed within W3C by the Media Fragment Working Group[1] (MFWG). Its mission is to address media fragments on the Web using Uniform Resource Identifiers (URIs). Although most HTTP streaming solutions rely on the use of regular HTTP 1.1 Web servers, we assume in this paper the availability of Media Fragments-aware servers for HTTP streaming and describe the impact of this availability for HTTP streaming solutions. Additionally, since the Media Fragments 1.0 specification also foresees a scenario for serving Media Fragment URIs using regular HTTP 1.1 Web servers, we will elaborate on how this scenario fits in the current HTTP streaming solutions.

2. MEDIA FRAGMENTS 1.0

The Media Fragments 1.0 specification supports three different axes for media fragments: temporal (i.e., a time range), spatial (i.e., a spatial region), and track (i.e., a track contained in the media resource). Since the spatial fragment axis is not relevant in the context of HTTP streaming, we will not further discuss it. Further, the specification recommends both

[1] http://www.w3.org/2008/WebVideo/Fragments/

the URI syntax and the protocol for the retrieval of Media Fragment URIs over HTTP [3].

2.1. URI Syntax

The specification defines the syntax for `mediafrag` within the URI `protocol://path/mediafile#mediafrag`. For brevity, we give a simple example for both the temporal and the track axis; the full syntactical details can be found in the specification [3].

- Temporal: `http://foo/media.mp4#t=10,30` identifies the time range [10s,30s[of media.mp4.
- Track: `http://foo/media.mp4?track=vid` identifies the video track of media.mp4.

Both URI fragments and URI queries can be used for media fragment addressing. Using a URI fragment means that the media fragment is a secondary resource and hence must be expressible in terms of byte ranges pointing to the parent resource. On the contrary, URI queries result in new resources, resulting in no restriction regarding the bytes used to represent the fragment. Note that, although track fragments can always be expressed in terms of byte ranges, the amount of byte ranges for a certain track is infeasible high when tracks are interleaved. Therefore, track fragments are typically addressed using URI queries when supported by the server or interpreted locally when the server is unable to extract the requested track (e.g., in case of a regular HTTP Web server). Local interpretation means that all tracks are downloaded by the UA, after which the UA picks the requested tracks. Temporal fragments are typically addressed using URI fragments.

2.2. Media Fragment Retrieval over HTTP

The current Web infrastructure, based on the HTTP protocol, is not aware of addressing methods other than bytes to point to a portion of a media resource. Therefore, in order to implement and deploy a system able to deal with Media Fragment URIs, the key requirement is to have a module that is able to translate media fragments (i.e., expressed in time or tracks) into fragments expressed in terms of bytes (i.e., byte ranges) [4]. Such a translation module can occur at the server or at the UA. The Media Fragments 1.0 specification describes a number of scenarios, based on the location of this translation module.

As specified in [5], fragment identifiers are separated from the rest of the URI prior to a dereference. In other words, they are not sent to the server and thus the identifying information within a fragment needs to be interpreted by the UA. Applying this to Media Fragment URIs, UAs must be able to parse and interpret media fragment identifiers. When the UA is able to perform the mapping between fragments and byte ranges, fragments can be requested in terms of byte ranges (using regular HTTP 1.1 byte range requests), as illustrated in Fig. 1.

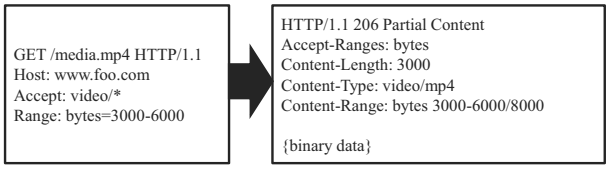

Fig. 1. UA-mapped Media Fragment retrieval.

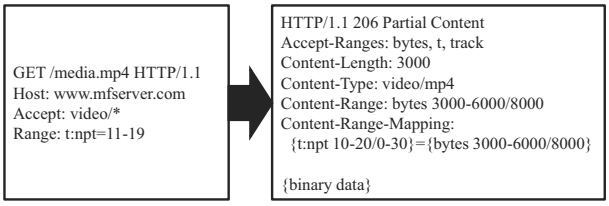

Fig. 2. Server-mapped Media Fragment retrieval.

In a second scenario, i.e., if the UA needs help to perform the mapping between media fragments and byte ranges, the media fragment identifiers need to be communicated in some way to a Media Fragments-aware server. Therefore, the MFWG recommends a protocol for retrieval of media fragments over HTTP. More specifically, a number of new HTTP headers were developed, allowing to provide media fragment information within an HTTP request. The details of the exact syntax can be found in the specification [3]; examples of these new headers are provided in Fig. 2. In this figure, a temporal range ([11s,19s[) is requested by using a time unit in the HTTP Range request header. The Media Fragments-aware server interprets the Range header, performs the mapping from time to byte ranges, extracts the requested bytes, and responds to the UA. As one can see, the HTTP response message contains a header (i.e., Content-Range-Mapping) indicating the actual extracted temporal range. The latter can differ from the original requested temporal range since random access points do not necessarily correspond to the range boundaries and the fragments returned by the server have to start with a random access point. The returned temporal fragment will always include the requested temporal fragment. Note that the UA can also request codec setup information, together with the temporal range (in the example, the Range header would then contain `t:npt=11-19;include-setup`). The response would then consist of an HTTP multipart response message, containing both the setup information and the bytes corresponding to the temporal range.

Finally, retrieving a track fragment using a URI query simply comes down to the download of a resource, as illustrated in Fig. 3. It is important to note that, when using URI queries and/or the newly defined HTTP headers for media fragment retrieval, the server needs to be a Media Fragments-

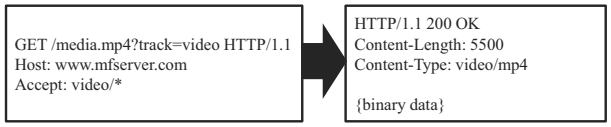

Fig. 3. Retrieving track fragments with a URI query.

Listing 1. Composing representations using MDP.

```
<MPD minBufferTime="PT2S" mediaPresentationDuration="
    PT30S" baseURL="http://example.com/">
  <Period start="PT0S">
    <Representation mimeType="video/3gpp; codecs=avc"
        bandwidth="256000">
      <SegmentInfo duration="PT30S" baseURL="media.3
          gp?track=highvid" />
    </Representation>
    <Representation mimeType="video/3gpp; codecs=avc"
        bandwidth="128000">
      <SegmentInfo duration="PT30S" baseURL="media.3
          gp?track=lowvid" />
    </Representation>
  </Period>
</MPD>
```

aware server, thus also containing a fragment-to-byte range translation module. Only in the first scenario, where the UA is able to perform the mapping, a regular HTTP 1.1 Web server is sufficient to serve the media content.

3. COMPOSITION OF MEDIA REPRESENTATIONS

As discussed in Sect. 1, HTTP streaming solutions make use of a manifest file. It may describe different representations (e.g., different bit rates, languages, or resolutions) of the same media content. Typically, these representations correspond to different media resources or track combinations within one media resource. The latter means that we can point to a representation in terms of a Media Fragment URI, using the track axis. This way, it is possible to store different representations within the same media resource.

Examples of manifest files using Media Fragment URIs to point to representations are shown in Listings 1 and 2, using the Media Presentation Description (MPD) and M3U8 syntax respectively. Different representations/versions are represented by means of different URI queries. For instance, `media.3gp?track=highvid` represents the high quality version.

When different representations correspond to different layers/views of a scalable/multiview media resource, the proposed approach will fail. More specifically, the current Media Fragments 1.0 specification does not provide explicit solutions for addressing scalability layers or alternative views. However, it should be noted that scalability layers and alternative views are very similar to tracks; the only difference is

Listing 2. Composing representations using M3U8.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=256
http://example.com/media.m3u8?track=highvid
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=128
http://example.com/media.m3u8?track=lowvid
#EXT-X-ENDLIST
```

Listing 3. Scalable media resources provide additional representations.

```
<MPD minBufferTime="PT2S" mediaPresentationDuration="
    PT30S" baseURL="http://example.com/">
  <Period start="PT0S">
    <Representation mimeType="video/3gpp; codecs=avc"
        width="352" height="288" bandwidth="256000">
      <SegmentInfo duration="PT30S" baseURL="media.3
          gp?track=0" />
    </Representation>
    <Representation mimeType="video/3gpp; codecs=svc"
        width="176" height="144" bandwidth="128000">
      <SegmentInfo duration="PT30S" baseURL="media.3
          gp?track=1_0" />
    </Representation>
    <Representation mimeType="video/3gpp; codecs=svc"
        width="352" height="288" bandwidth="280000">
      <SegmentInfo duration="PT30S" baseURL="media.3
          gp?track=1_1" />
    </Representation>
  </Period>
</MPD>
```

that the former can be dependent on other layers/views while this is not the case for the latter. Thus, if these layers/views are identifyable, the track axis could be used to address them.

Consider a media resource containing two tracks: an H.264/AVC video track and an SVC video track with two spatial layers. The corresponding MPD manifest is depicted in Listing 3. As one can see, each scalability layer corresponds to a representation (which is similar to what MPEG DASH will support). Further, the two spatial layers are identified through the track axis (e.g., `media.3gp?track=SVC_layer0` could refer to the spatial base layer of the SVC track). Of course, this only works if the server disposes of an SVC bitstream extractor and is aware of the mapping between layer identifiers (e.g., `SVC_layer0`) and scalability layers.

4. HTTP STREAMING USING THE MEDIA FRAGMENTS PROTOCOL

In typical HTTP streaming scenarios, not only the different representations of media content are described in the manifest, but also information regarding the structure of one representation. More specifically, for each representation, different segments or temporal fragments are described. These dif-

ferent segments can also be represented by temporal Media Fragment URIs. Moreover, UAs could even avoid segment information and compose their own temporal Media Fragment URIs. The compact manifest shown in Listing 1 would then be sufficient for a UA to retrieve the media content using HTTP streaming.

Based on the manifest, the UA can start interpreting or generating Media Fragment URIs, each representing one temporal piece of a certain representation. These Media Fragment URIs will be translated into HTTP Range requests containing using a time unit (see also Sect. 2.2). We illustrate the behavior of the UA by using the manifest of Listing 1.

The UA decides to request the high quality representation (i.e., the media resource `media.3gp?track=highvid`). The UA can choose the target duration of each segment, for example 3 seconds. Consequently, the first segment corresponds to the Media Fragment URI `http://example.com/media.3gp?track=highvid#t=0,3`, which results in the following HTTP request:

```
GET /media.3gp?track=highvid HTTP/1.1
Host: www.example.com
Accept: video/*
Range: t:npt=0-3;include-setup
```

The UA adds the 'include-setup' parameter to the Range header in order to retrieve the codec setup information. Note that the latter can be seen as an initialisation segment in 3GPP's Adaptive HTTP Streaming specification. The Media Fragments-aware server interprets the request, calculates which bytes from the requested resource need to be returned, and constructs the following HTTP response message:

```
HTTP/1.1 206 Partial Content
Accept-Ranges: bytes, t
Content-Length: 100
Content-Type: video/3gpp
Content-Range-Mapping:
  {t:npt 0-3.6/0-30;include-setup}=
  {bytes 0-8,9-99/1067}
Content-type: multipart/byteranges
--SEP
Content-type: video/3gpp
Content-Range: bytes 0-8/1067
{binary data}
--SEP
Content-type: video/3gpp
Content-Range: bytes 9-99/1067
{binary data}
--SEP--
```

The response of the server consists of a multipart message containing the codec setup data and the bytes corresponding to the requested time range. However, due to random access point boundaries, the server returned bytes corresponding to the time range [0s,3.6s[, as indicated by the Content-Range-Mapping header. This means that the next request of the UA will correspond to the Media Fragment URI `http://example.com/media.3gp?track=highvid#t=3.6,6.6`, without requesting codec setup information since this is already retrieved.

After retrieving the bytes from 0s to 16.2s, the UA decides to change the representation because less bandwidth is available. The following HTTP request is sent to the server:

```
GET /media.3gp?track=lowvid HTTP/1.1
Host: www.example.com
Accept: video/*
Range: t:npt=16.2-19.2;include-setup
```

The UA requests bytes from the lower quality version (indicated by the track parameter) and requests new codec setup information. Since bytes up to timepoint 16.2s were already retrieved (high quality), the UA seeks to position 16.2s in the low quality version. The returned HTTP response message contains the following Content-Range-Mapping header:

```
{t:npt 15.9-19.3/0-30;include-setup}=
                    {bytes 0-7,308-361/534}
```

Since random access points are not aligned between the two representations, the server returns bytes corresponding to the underlying random access boundaries (i.e., time range [15.9,19.3[). Thus, the UA can seamlessly switch from high to low quality between 15.9 and 16.2 seconds.

The presented approach lets UAs determine how fine or coarse the requested segments are in terms of duration. Also, the UA does not have to discover the location of random access points within the representation and their segments. Indeed, the server is able to perform the segment extraction and communicates the random access point boundaries to the UA. Also, codec initialisation information is determined by the server and requested by the UA through the 'include-setup' parameter. Further, live scenarios are also supported by Media Fragment URIs by using wall-clock time codes in the temporal axis. `#t=clock:2010-10-11T11:19:01Z` for example is a temporal fragment starting on 11th Oct 2010 at 11hrs, 19min, 1sec. This way, not only information regarding the different segments within a manifest file is avoided, also updates of the manifest necessary in live scenarios are not necessary anymore thanks to the use of wall-clock time codes.

An HTTP adaptive streaming solution based on Media Fragment URIs as the one presented in this paper looks promising. It only requires a limited description of the different representations in a manifest and does not impose any restrictions to underlying media formats. However, the presented solution only works if the W3C Media Fragments 1.0 specification is implemented within the Web infrastructure. More specifically, Web servers need to be extended with support for the newly introduced Range unit (i.e., time) and HTTP headers, as well as with a media fragments extractor module that is able to perform the translation between media fragments and byte ranges. Additionally, current HTTP caches will not be able to cache media fragments as they are

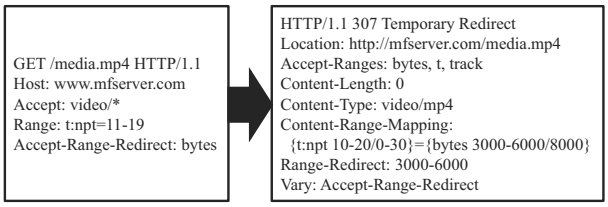

Fig. 4. Cacheable media fragments.

not aware of the new Range units. Therefore, specialized media fragment caches should be developed in order to cache media fragments served by Media Fragments-aware servers.

5. AVOIDING MEDIA FRAGMENT-AWARE SERVERS

The MFWG does recognize that there should be solutions for serving and retrieving media fragments using the current Web infrastructure. For instance, the UA can request the server to perform the translation between media fragments and byte ranges, after which the UA uses the obtained byte ranges to perform regular HTTP 1.1 byte range requests (HTTP communication is illustrated in Fig. 4). However, this solution still requires a Media Fragments-aware server. Such a server can be avoided if the UA is able to perform the mapping between media fragments and byte ranges without help from the server, as discussed in Sect. 2.2.

The translation between media fragments and their corresponding byte ranges is dependent on the underlying media container. Generally, two approaches exist to perform a remote[2] translation, dependent on the organization of the container format.

When the underlying container format of the media resource supports a full index providing a complete mapping of time and byte-offsets, then only the first couple of bytes corresponding to the index need to be downloaded. Subsequently, the index is interpreted by the UA in order to calculate the mapping between media fragments and byte ranges. The latter is dependent of the container format since different container formats use different structures to represent the index. Examples of container formats providing support for such a full index are MP4 and ASF.

When no full index is provided at the beginning of the media resource, the proper byte positions need to be found for a given media fragment identifier. This is obtained by applying a bisectional search over HTTP. More specifically, the UA starts by guessing which byte position corresponds to a given temporal position. Subsequently, these bytes are retrieved and interpreted. If the byte position is too high/low, another guess is made in the right direction until the correct byte offset is found. It is clear that this method is less efficient in terms of HTTP round-trips than the first method. Examples of container format structures where bisectional search over HTTP could be applied are Ogg files, WebM files, and fragmented MP4 files.

[2]Note that 'remote' indicates that the mapping is calculated without having the full media resource at our disposal.

Listing 4. M3U8 composition served by NinSuna (/Media/Apple/Avatar/Teaser.m3u8).

```
1  #EXTM3U
   #EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=296960
   /Media/Apple/Avatar/Teaser.m3u8?track=1;2
   #EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1230848
5  /Media/Apple/Avatar/Teaser.m3u8?track=3;4
   #EXT-X-ENDLIST
```

6. IMPLEMENTATION WITHIN APPLE'S HTTP LIVE STREAMING

In order to evaluate the feasibility of integrating Media Fragment URIs into HTTP adaptive streaming techniques, we implemented the above described ideas into Apple's HTTP Live Streaming solution [6]. More specifically, we used M3U8 as format to describe the manifest information. Also, native players supporting HTTP Live Streaming such as iPod/iPad/iPhone and QuickTime X work with the presented solution.

As a server solution, we used NinSuna[3], which is a fully integrated media adaptation and delivery platform supporting the Media Fragment URI 1.0 specification [7]. Moreover, NinSuna provides support for both query and fragment-based media fragment retrieval along the temporal and track axis. Note that the examples in the listings below are available online for testing purposes (base URL is `http://ninsuna.elis.ugent.be`).

As discussed in Sect. 3, different representations of the same media content can be represented in terms of media fragment URIs, using the track axis. This is illustrated in Listing 4, where two representations are described. Tracks '1' and '2' correspond to the low quality audio and video version, while tracks '3' and '4' correspond to the high quality version.

When the UA chooses one representation to start the playback (e.g., the low quality version), it requests the corresponding manifest (see Listing 5). Since we use an existing, non-modified HTTP Live Streaming UA, we need to explicitly list the media fragment URIs of the different time segments. The latter are expressed with media fragment URIs using the time and track dimension.

However, there is a difference in the approach explained in Sect. 4. Since HTTP Live Streaming UAs such as the iPhone do not support the Media Fragments URI protocol

[3]`http://ninsuna.elis.ugent.be`

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.