Dynamic Adaptive HTTP Streaming of Live Content

Thorsten Lohmar[♣], Torbjörn Einarsson[♦], Per Fröjdh[♦],Frédéric Gabin[♥], Markus Kampmann[♣]

[♣]Ericsson GmbH
52134 Herzogenrath, Germany

[♥]Ericsson France
91300 Massy, France

[♦]Ericsson AB
164 80 Stockholm, Sweden

[Thorsten.Lohmar | Markus Kampmann | Frederic.Gabin | Per.Frojdh | Torbjorn.Einarsson] @ ericsson.com

***Abstract*— MPEG has recently released a first public draft of the new Dynamic Adaptive Streaming over HTTP (DASH) specification. The new streaming technique is based on the 3GPP and partly also on the Open IPTV Forum (OIPF) specifications. In this paper, we give an overview about the new DASH specification with a special focus on Live streaming services. In order to re-use existing web content distribution schemes, the new streaming technique provides the live stream as a sequence of files, which are continuously downloaded by the streaming client. This way of streaming introduces new delay components into the system streaming. Live Streaming technology is often used for events like sport events to allow other users to *virtually* participate. It is generally preferred to minimize the end-to-end delay for live services. In the paper, we identify and analyze the different delay components of the new adaptive HTTP streaming technique and how they contribute to the end-to-end delay of live services. We also discuss the dependencies and system implications when minimizing the end to end delay. The evaluation principles are also applicable for other adaptive HTTP streaming formats.**

Keywords- Adaptive HTTP Streaming; DASH; AHS; OIPF HAS

I. INTRODUCTION

With the increasing data rates offered by fixed and mobile networks, multimedia streaming services are getting wider distribution [1][2]. The mobile video consumption is stimulated with the increase number of Smartphones in the market. Smartphones *invite* for multimedia consumption with its larger screen sizes, higher resolutions and improved usability. However, content providers need to offer their content also in different formats and via fixed accessed in order to reach the whole set of Internet connected devices like laptops and also Set-top-Boxes.

In order to realize video streaming, the Real-Time Streaming Protocol (RTSP) [3] and Session Description Protocol (SDP) [4] for session setup and control and the Real-Time Transport Protocol (RTP) [5] for the transport of real-time speech, audio and video are widely used e.g. in the 3GPP Packet-switched Streaming Standard (PSS) [6][7][8] specified for multimedia streaming over mobile networks. In order to cope with fluctuating bitrates on the transmission path, adaptive RTP streaming techniques are used [9]. However, RTSP/RTP streaming has problems in case of firewalls and NAT traversals, and requires dedicated network infrastructure that cannot be used for other web content.

978-1-4577-0351-5/11/$26.00 ©2011 IEEE

Therefore, alternatives to RTSP/RTP streaming are getting popular which are based on the Hypertext Transfer Protocol (HTTP) [10].

In the last 2 years, a number of proprietary solutions for adaptive HTTP streaming for live and on-demand content have appeared on the market. Adaptive streaming is required for IP networks, because the needed bandwidths for high definition and high bitrate streaming is not in place for all different access networks. Furthermore, since some network links are often shared resources, the available bandwidth may change at any time during a session.

3GPP decided in January 2009 to open a Work Item to define a HTTP streaming standard that would allow for dynamic bitrate adaptation and improved live services with 3GP files. The standard, now known as 3GP-DASH (Dynamic and Adaptive Streaming over HTTP) covers the main aspects of HTTP streaming including storage, transport and media rate adaptation to available link bitrates. 3GP-DASH was first introduced in 3GPP Release 9 of the PSS standard [6][7], finalized in March 2010 as Adaptive HTTP Streaming (AHS). The standard has been adopted by MPEG as a baseline for MPEG-B DASH which is currently under development.

This paper focuses on particular issues with *Live* adaptive HTTP streaming. Live streaming has a number of requirements and challenges compared to on-demand streaming, which require special considerations.

The outline of the paper is as following: Section 2 introduces the general features and mechanisms of the Dynamic Adaptive Streaming over HTTP (DASH) specification. Section 3 describes the general functionalities and principles of HTTP streaming of live content as well as its challenges. In Section 4, a delay analysis of Live DASH streaming is given where the additional delay introducing components are presented, the total delay is estimated and a comparison with RTP streaming is carried out.

II. I. DASH OVERVIEW

The DASH standard is composed of two main parts. One part defines the Media Presentation Description (MPD) that is used by the client to find the links to access the content. The other part defines the format of the content in terms of media segments as extensions to the 3GP and MP4 file formats. The default protocols of media segments retrieval is HTTP although other protocols which use URLs to uniquely identify media segments may be used as well.

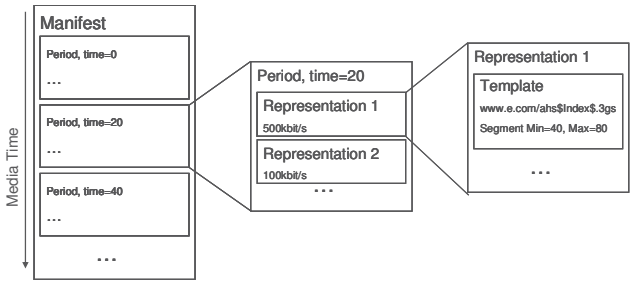

Figure 1. MPD layout

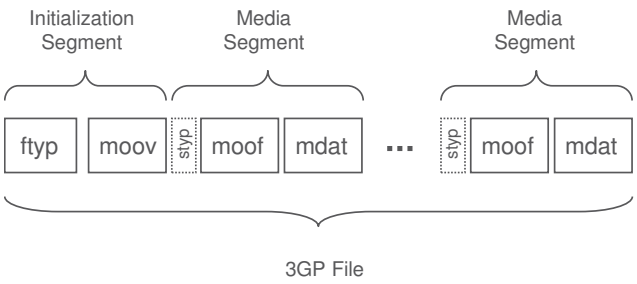

Figure 2. 3GP based HTTP streaming segments

A. *The Manifest File (MPD)*

The purpose of the MPD is to give location and timing information to the client to fetch and playback the media segments of a particular content. The MPD syntax is defined in XML. Typically the MPD file is fetched using HTTP at the start of the streaming session, but for flexibility, the MPD may also be updated at a minimum time interval.

The MPD consists of three major components, namely Periods, Representations and Segments. As depicted in Figure 1, Period elements are the outermost part of the MPD. Periods are typically larger pieces of media that are played out sequentially. Inside a Period, multiple different encodings of the content may occur. Each alternative is called a Representation. These alternative Representations can have, for example, different bitrates, frame rates or video resolutions. Finally, each Representation describes a series of segments by HTTP URLs. Those URLs are either explicitly described in the Representation (similar to a playlist) or described through a template construction, which allows, the client to derive a valid URL for each segment of a representation. The MPD format is flexible and can support other media container formats such as MPEG2 TS. Content playlist or ad-insertion functionality can easily be achieved by chaining periods of different content.

B. *3GP and MP4 Media Segments*

The 3GP file format is based on the ISO base media file format. A 3GP segment for HTTP streaming is either an initialization segment or a media segment (see Figure 2). An initialization segment contains configuration data (formatted as 'ftyp' and 'moov' boxes of the file format), whereas a media segment is a concatenation of one Segment type box ('styp') and one or more movie fragments of media pointers and samples ('moof' and 'mdat' boxes). Concatenation of the initialization segment and one or more media segments of the same representation results in a valid 3GP file.

The 3GP file format was extended for the specific HTTP streaming requirements Except the new 'styp' box that is similar to the 'ftyp' box, but discarded by old readers, there are new 'sidx' and 'tfad' boxes.

The Segment Index box ('sidx') provides relative timing information with respect to the period start for time recovery after seeking. It also provides a list of random access points, but that information can also be conveyed as a sample flag. The Track fragment adjustment box ('tfad') on the other hand provides relative timing information between the tracks for media synchronization. Both 'sidx' and 'tfad' are optional.

Note one key principle of the new adaptive HTTP streaming is that media segments are identical to all users; adaptivity is obtained simply by switching between segments of alternative representations. This property makes 3GP-DASH HTTP cache and Content Delivery Network (CDN) friendly. The media segments, uniquely identified by its URL can be served from intermedia HTTP Proxy/Caches in the same way as any other Web Content.

Finally, the codecs recommended for Adaptive HTTP Streaming in 3GPP are identical to those defined for other streaming solutions such as 3GPP PSS streaming.

C. *MPEG-TS Media Segments*

After finalization of the 3GPP Rel-9 specification of adaptive HTTP streaming, the Open IPTV Forum (OIPF) subsequently adopted this specification as part of their Release 2 which was published in September 2010 [11]. The MPD syntax and semantic is re-used with some profiling. An extension to support media segments in MPEG2 Transport Stream (MPEG2-TS) format was also specified.

The MPEG2-TS extensions of OIPF are very much aligned with the 3GP media segments. The MPD indicates through MIME types that the format of the media segments is MPEG2 TS. The mimetypes "video/mpeg" and "video/mp2t" are allowed.

The OIPF MPEG-2 TS format is a subset of the full TS format. This makes it possible to create a compliant MPEG2-TS stream by concatenating the media segments fetched by the client.

The Program Specific Information (PSI) tables may be either contained in an initialization Segment or in the media segments. Each media segment must start with a random access point. All media representations are further time aligned to simplify switching and bitrate adaptation. It is further recommended to use the MPEG2-TS *random_access_indicator* fields to signal random access points within media segments, thereby simplifying trickplay and seeking operations on the client side.

In parallel to this work in Open IPTV forum, MPEG issued a call for proposal on HTTP streaming and reviewed the various proposals in July 2010. Among the responses was one proposal to adopt the 3GPP Adaptive HTTP streaming solution as baseline in the MPEG Working Draft.

This proposal was adopted and MPEG named the overall solution as DASH. Since then, 3GPP agreed to use the MPEG defined name of DASH in its Rel-10 3GP-DASH specification TS 26.247 [12].

In September 2010 the MPEG DASH Ad hoc meeting agreed to add the Open IPTV Forum MPEG2-TS extension in their baseline. MPEG will further enhance their Working Draft by defining the MP4 file format for HTTP streaming segments and add the necessary extensions to support MPEG codecs like e.g. MVC. MPEG DASH specification is expected to be published in the end of 2011.

DASH specifications are now available for server and client implementers with 3GP and MPEG2-TS file formats. The IMTC Forum [13] organizes a test fest event in October 2010 involving several vendors to perform interoperability tests of DASH clients and servers. We expect MPEG-DASH to continue to be fully compatible with 3GP-DASH and the MPEG2-TS OIPF extensions while providing necessary support for MP4 files and other enhancements.

III. LIVE STREAMING

The phrase *Live streaming* typically refers to a continuous stream from a live event. Traditional broadcasted TV from live events is a good example of such a service. The end-to-end delay experienced in particular in the content distribution branch should be minimized, so that the end-user can attend the event in real time. However, some delay in order of a few seconds is not avoidable due to encoding and media transport.

In traditional streaming or broadcasted TV, the content is continuously pushed to the client. The client receives a continuous flow of data. In the modern adaptive HTTP streaming cases, it is the client that pulls the content as a sequence of files called media segments from a server. This pulling and also the segmentation of the content into a sequence segments makes the delay even longer.

The end-to-end delay is less critical when consumers are not able to compare different delivery channels. But when consumers can compare different reception path, then the preferred viewing devices should provide the lowest delay. Especially for live sports, it may be very annoying to hear neighbors screaming seconds before you see a goal on your screen. Therefore, it is important to understand the contributing delay components and how to reduce the end-to-end delay.

This section is structured as following. We start describing the expected client behavior for adaptive HTTP streaming with special focus on DASH. After that, we focus on the server side segmentation process and how the server makes the segments available. Then we describe the MPD update procedure for DASH, which is needed to allow advertisement insertion during a Live Session.

A. *Expected Client Behavior for On Demand and Live*

As mentioned above, DASH operates by segmenting the continuous stream into a sequence of segments. The segments are distributed as independent files over the network. The client fetches the sorted sequence of files and concatenates them back into a continuous media stream. Clients know the media segments URLs and its sequence from the Media Presentation Description (MPD).

Figure 3 depicts the media retrieval procedure, which is applicable for Live and On-Demand. The following steps are executed at the client in order to receive a continuous service:

1. The client first fetches the MPD from a server. The MPD is identified by a URL and describes the media type (on-demand or live), the available media representations (media qualities) and the URLs of the media segments.
2. The client processes the MPD and selects one suitable representation. The representation should be of low media quality in order to allow a fast start-up of the streaming session.
3. The client starts requesting segments. In case of Live sessions, the client must carefully determine the URL for the segment to select. This is discussed later in detail.
4. The client measures the download rate of the media segment. The client knows the media data duration contained in the media segment and also the file size. The client should compare the download duration of the media segment against the contained media data on the segment. The client can change the media quality according to the estimated available link bitrate by selecting a different representation.
5. The client buffers content as recommended by the MPD before rendering the content.
6. Content is played out and Segments are continuously requested.
7. The client should frequently update the MPD when this is indicated in the MPD. DASH does not require updating the MPD for live sessions. This is described later in detail.

Figure 6 depicts an example MPD instance for Live DASH streaming.

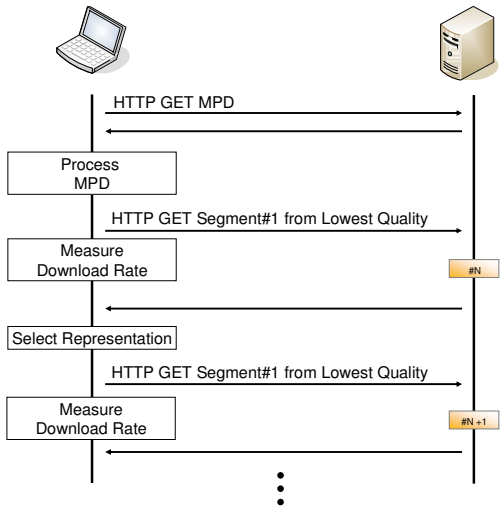

Figure 3. Media reception sequence

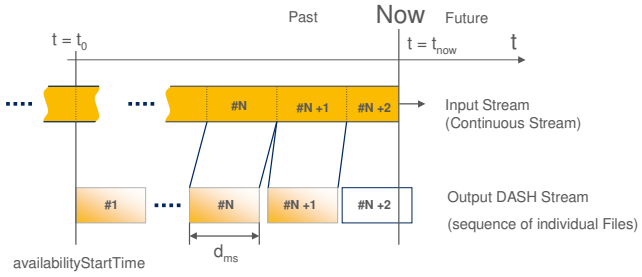

Figure 4. Segmentation of a continuous live stream into a sequence of files

B. *Preparing Live Content*

In case of Live session, the sequence of media segments are created on the fly from a continuous media stream. Figure 4 depicts this segmentation process of a live session for a single quality representation. The upper stream is the input stream, which is received by a media segmenter function at $t = t_{now}$. The lower stream shows the created sequence of the media segments. Media segment #1 is the oldest media segment and was created at the beginning of the live session. Media segment #N+2 is not yet available on the HTTP server, although the starting time $t = t_{n+2}$ of the media segment is already passed. The segmenter function creates a new media segment every d_{ms} seconds. Thus, each media segment contains here d_{ms} seconds worth of media data.

In order to understand and minimize the end-to-end delay, we need to understand the segmentation process and when new media segments can be available on the server.

Let's number the sequence of media segments of a live session with indexes i >= 1. Let t_0 be the time, when the segmenter receives the first bits of the continuous input stream (the upper streaming in Figure 4). The index i of the first media segment is 1.

The index of the latest completely created media segment is calculated as

$$i = floor\left(\frac{t_{now} - t_0}{d_{ms}}\right) \qquad \textbf{(1)}$$

where t_{now} is the current live point and d_{ms} is the duration of media data per segment (cf. Figure 4). A segment is completely created when the segmenter has received all needed bits for the media segment from the input stream.

The media segment URLs may be described in template or in playlist forms. When the URLs are in playlist form, the receiver can regard the list of URls as an array and i as the index into the array (starting at value one).

One key issue for receivers of a DASH Live streaming session is to find the current "live point" (i.e. $t = t_{now}$). In case of traditional Live Streaming, the time "NOW" is associated with the reception of the first bits of the content stream. The client connects to the stream and the server starts pushing data from "NOW" onwards.

Other adaptive HTTP streaming techniques such as Apple's HTTP Live Streaming only describe available media segments in the m3u8 playlist, which is similar to the MPD. The DASH MPD describes all available and not-yet-available media segments either for the entire live session or up to the next MPD update. Thus, the DASH client need to determine available media segments by itself.

The client reads the start time of the live stream t_0 from the MPD. The value of the MPD element *availabilityStartTime* (see next section) gives the earliest availability time ($t_0 + d_{ms}$) of the first media segments and allows synchronization of the DASH segment stream with the wall clock time. If the client is properly time synchronized, it can calculate the latest available media segment on the server given the signaled (average) segment duration.

C. *Example MPD*

The Media Presentation Description is defined in XML. In Figure 6, an example MPD instance for a live service is depicted. Key MPD fields for live services are the type, the *minimumUpdatePeriodMPD* fields and further the *availabilityStartTime* and *availabilityEndTime* fields. The *minimumUpdatePeriodMPD* field gives the duration for MPD updates. The client should expect that the description or locations of future media segments is changed. For instance, a new period element with a lower number of media quality representations may be added.

The *availabilityStartTime* is mandatory for live content. It associates a global UTC time with the start time for the first period in the MPD. Thus, it associates a global UTC time to the first media segment of the first period in the MPD. The optional element *availabilityEndTime* informs the client about the end of the live event. Media segments may not be available anymore after this time. This example MPD in Figure 6 describes all media segments between the start of the live session and the end. All media segments are assumed to be hosted by the same server in the example MPD, since the media segments are described relative to the location of the MPD.

When fetching media segments, the DASH client must carefully determine the URL of the latest available media segments (the Live point $t = t_{now}$). The client must be time synchronized with the server and determine the Live-point by summing up the media segments.

The MPD indicates that the live Streaming session starts at $t_0 + d_{ms}$ = 2010-10-11T13:50:44Z (cf. MPD instance below). The average media segment duration for the media segments is indicated as d_{ms} = 10 sec. The min buffer time is also indicated as 10 sec (thus, one segment shall be buffered).

At time $t = t_{now}$ (cf. Figure 4), clients should have calculated the latest available media segment at time of joining to be i = N+1 and starts fetching the media segment i-1 = N to fulfill the buffering constraints.

The content provider may remove media segments before the Live session ends from the server e.g. to save storage capacity. This server behavior is expressed through the MPD

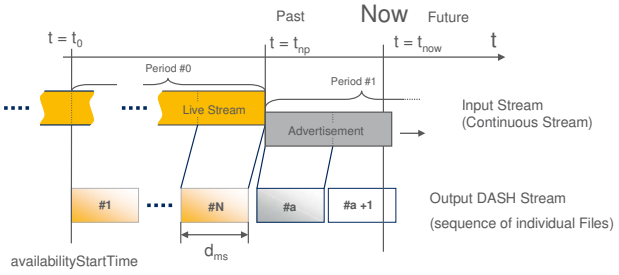

Figure 5. Advertisement insertion

element *timeShiftBufferDepth*. It describes the availability duration of media segments on the server. The client is in timeshift mode when it does not render the latest available media segment given buffering constraints.

In case the MPD contains the *minimumUpdatePeriodMPD* element, the DASH client should check for MPD updates at the given update interval. The MPD and its updates are fetched using the same URL. The content provider may add periods to the MPD, for instance to add an advertisement or another content item. The MPD update period may be longer than the media segment duration. For instance, the media segments may be of average duration of 10 seconds, whereas the client updates the MPD every minute. The MPD for Live cases may contain future media segment URLs, which are not yet available. The client needs to determine the latest available media segment using the *availabilityStartTime* and the current time of day.

D. *Ad Insertion*

Advertisements and commercials are very common in today's world. Services can become often cheaper or even free, when the services are financed through advertisements. The new DASH standard allows two methods to insert advertisements into live streams

- Embedding the advertisement transparently into the stream
- Adding the advertisements as new period into the MPD

The first method of transparently inserting advertisements into the steam is well known from Broadcast TV. The new content is seamlessly integrated with the earlier content stream and the presentation and decoding timestamps are increasing monotonically with the correct increments.

The second method requires frequent updating of the MPD by renewing the MPD from the given URL. The client is made aware of this by the presence of the *minimumUpdatePeriodMPD* attribute, which value describes the minimal period. The client should expect that new periods are added, which may use different URL constructors.

Figure 5 depicts the schematics of advertisement insertion by adding a new period in the MPD. The first period (period #0) is closed (start and end time defined) and no additional media segment URLs are added to this period. In case of a template based URL description, the *endIndex* attribute is present. The new period (period #1) is active.

Each Period element in the MPD has a start attribute, which describes the start time of this period. Terminals use the representations and URL constructions from the new period from $t = t_{np}$ onwards. The number of representations may change. For instance the advertisement may not be available in HD, while the main content is available in HD. Further, the media segments may also be served from a different Web Server with different codec parameter settings. Even different codecs may be used at period breaks.

This procedure is not limited to advertisement insertion into live streaming. It can be used to change from one stream to another. A provider may concatenate different streams without re-multiplexing them to the same timeline.

IV. DELAY ANALYSIS OF LIVE DASH STREAMING

As mentioned above, especially live sports may require short end to end delay. However, this comes at an overhead cost. A service provider should therefore find the right balance between delay and overhead, depending on the content. In this section, we evaluate the end to end delay of DASH live streaming service. It is calculated slightly different than for other live streaming formats, since the MPD also describes "not-yet-available" media segments.

```
<MPD xmlns="urn:3GPP:ns:PSS:AdaptiveHTTPStreamingMPD:2009" availabilityStartTime="2010-10-11T13:50:44Z"
 availabilityEndTime="2010-10-11T16:50:44Z" timeShiftBufferDepth="PT1M0.00S"
 type="Live" minBufferTime="PT10.00S">
<Period segmentAlignmentFlag="true">
   <Representation bandwidth="128000" mimeType="video/3gpp" startWithRAP="true">
     <SegmentInfo duration="PT10S" baseURL="mt500">
     <InitialisationSegmentURL sourceURL="fifa128seg_i.3gp"/>
     <UrlTemplate sourceURL="$Index$.3gs"/>
     </SegmentInfo>
   </Representation>
   <Representation bandwidth="512000" mimeType="video/3gpp" startWithRAP="true">
     <SegmentInfo duration="PT10S" baseURL="mt1000">
     <InitialisationSegmentURL sourceURL="fifa512seg_i.3gp"/>
     <UrlTemplate sourceURL="$Index$.3gs"/>
     </SegmentInfo>
   </Representation>
</Period>
</MPD>
```

According to the DASH standard [6][13], the feature of Updating MPDs is not required for Live streaming. Thus, a live stream may be described by a single, not-updating MPD as given in Example MPD instance.

The main contributing delay components are

1. Content acquisition and preparation
2. Server-side packetization of media segments (segmentation delay)
3. Asynchronous fetch of media segments
4. Time to actually download the segments
5. Buffering at client side
6. Decoding and playout in the client

Points 1) and 6) are not specific for HTTP streaming, and not further discussed here. In the following, we will discuss how each of the above mentioned components contributes to the end-to-end delay.

A. *Segmentation Delay*

The server-side packetization is specific for segment-based streaming, and depends directly on the media segment duration (dms in Figure 4).

For ISO-file based segments, there is an unavoidable packetization delay, since the moof box cannot be constructed until all the media sample sizes are known. This means that the server must buffer all media for one fragment duration, which leads to a minimal server-side packetization delay equal to the media fragment duration. For example, the media segment N+2 in Figure 4 is not yet available, since the server has not yet received the full media segment duration d_{ms}.

However, by making a similar optimized packetization as for progressive download, the decoder could unpack and decode the media segments while downloading. Recall, that the different boxes needed to decode the media samples of a progressive download file are in the moov box at the beginning of an MP4 file, so that the receiver gets those boxes first. Further, audio and video media data are interleaved in the MP4 file, so that the client gets the needed media data at the same time.

In the case of media fragments, the interleaving is done by using trun boxes. .

For MPEG-2 TS based media segments, there are no length fields in the file format, so the stream could be piped directly into an HTTP response as long as the content-length does not need to be specified. However, if one wants to use standard HTTP servers, it is still better to make separate files for each segment and serve them as static file content. Then, again, this results in a minimal packetization delay equal to the media segment duration. $T_{ssp} = d_{ms}$.

B. *Asynchronous fetch of media segments*

The asynchronous fetch of the segments is due to that there is no synchronous signaling from the server that a new segment is ready, but the client must poll for data. The client determines the latest available media segment as described in Section 3.2. Media segments of the same representation should have approximately the same duration, but apart from duration variations some segments may also take longer to encode than other. Thus, there is no guarantee that the segmenter makes a media segment available at precisely duration after the latest segment.

The client asynchronously makes an HTTP GET request as more data is needed. The client could be very aggressive and poll repeatedly to find the latest segment as soon as it is available but this is an extra overhead, and load on the server. To avoid unsuccessful fetches, the client needs to schedule the fetches some time period after the calculated ideal availability time, leading to a typical delay contribution $T_{afs} = d_{ms}$.

C. *HTTP download time*

The HTTP download time does of course also matter. The download time is mainly determined by the media segment size (in bytes) and the available link bitrate (in bps), and the link delay. In the best case, then download time is only a fraction of the play-time of the segment. In the worse case the download time is just as long as the playtime of the segment. If the download time gets longer than the play time, or is foreseen to be longer, the client should switch to a lower bitrate representation. If the available bandwidth is used to a reasonable share, the download time of a media segment should be close to the segment duration time: $T_{hd} = d_{ms} + d_{link}$.

D. *Buffering in the client*

The buffering in the client is needed to provide a smooth media play-out, and hide transport jitter such a varying download time. Further, from the adativity point of view, the client side buffer should be sufficiently large, so that the client can interrupt the download of a media segments and still get a media segment of a lower quality representation before a buffer underrun occurs. The network bandwidth may suddenly decrease, leading to download time increase. The closer the media bitrate of the selected representation is to the available link bitrate, the more the client is exposed to changing network conditions (cf. Section 4.3).

To be robust towards network bandwidth fluctuations, a minimal target is that the client always has one full media segment in the buffer. To be more robust, it may be good to increase the buffer to 2 segments, leading to $T_b = 2\ d_{ms}$. Of course, this is an optimization done for minimal delay of live content. For stored content, it is advantageous to increase the robustness even further by having more segments buffered.

E. *Total delay and comparison with RTP*

Our estimate of the total minimal delay for HTTP streaming is thus

$$T_{tot} = T_{ssp} + T_{afs} + T_{hd} + T_b = 5d_{ms} + d_{link}, \qquad \textbf{(2)}$$

of which 2dms is buffering. Given this formula, it remains to choose a value for dms. The optimal value depends on the type of event, and in particular on the

competitive landscape. Receivers of an HTTP streaming session should not see the goal much later than people using broadcast TV, or similar.

Buffering and link delay is present also for RTP streaming, so in a direct delay comparison the main delay difference is $3d_{ms}$ - d_{RTP_ps}. Here, d_{RTP_ps} is the RTP packet serialization time, which is only some 10s of ms, e.g. 20ms for a link of 160kbps and packet size of 1000bytes. With a segment size of 1s, there is therefore roughly 3 seconds more delay in a live HTTP streaming service, compared to the corresponding RTSP/RTP-based service.

F. Overhead versus delay

Another issue that is worth to consider is the overhead that comes with smaller segments. For RTP, the total transport overhead for IPv4 is typically 40bytes * 40packets/s ~= 12kbps.

For HTTP streaming, we have an overhead of each HTTP request which is IP/TCP header (40 bytes) + the HTTP response header (size depends on URL etc but is typically more than 100 bytes). In addition to this, we have the overhead from the iso-file container. This consists of two parts; the first one is a fixed amount which for two tracks is around 200 bytes. The other part is directly proportional to the number of samples and depends on how many fields differ between each sample (size, duration, flags, etc). In a typical case there is 8 bytes per video sample (i.e. per frame) and 4 bytes per audio sample (typically 1024 audio samples in the case of AAC) . Assuming 30 frames/s and 24kHz sampling, this leads to 28kbps/segment + 3kbps which gives the data in the table below.

Packetization	Total overhead
HTTP segment duration = 1s	31 kbps
HTTP segment duration = 2s	17 kbps
HTTP segment duration = 10s	6 kbps
RTP	12 kbps

Thus, for lower media bitrates (like 100kbps), short segments give a substantial overhead, which should be taken into consideration.

V. SUMMARY AND CONCLUSIONS

In this paper we review recent efforts to standardize solutions for HTTP Streaming. The basis for these efforts comes from 3GPP which has defined a solution where a client downloads segments of 3GP (MP4) files given alternatives and directions in a Media Presentation Description. With this and support for MPEG-2 Transport Streams, added by OIPF, MPEG is developing an International Standard which will be finalized in July 2011.

The main reasons for using HTTP rather than a dedicated streaming protocol such as RTP/RTSP are that one can use standard HTTP servers, caches, and Content Distribution Networks, and also avoid problems with firewalls and NAT traversal. However, using HTTP for real-time streaming also requires careful consideration since a protocol designed for file delivery is not necessarily optimized for real time performance. Bandwidth fluctuations, in particular for mobile systems, require bitrate adaptation. This is solved by letting a client actively download and switch between suitable media segments rather than downloading an entire file.

The main focus of this paper is adaptive HTTP streaming of live content which poses further constraints in order to maintain low delays. Finally, the implications of using HTTP instead of RTP/RTSP for live streaming are analyzed. The best performance in terms of reduced delay is obtained with short media segment. However, even with durations as short as 1 second there will be an additional delay of around more than 2 seconds compared to RTP. Using a short segments also lead to a substantial increase in server load, so it should only be done if short delay is really important for the content at hand.

In general, it is not possible to optimize everything to the last microsecond in order to minimize the end-to-end delay. Media segments contain only average media duration, thus the client must be prepared that the segment is not yet available when fetching it. Clients and server (e.g. in the *availabilityStartTime*) should to some add small margins to secure that media segments are always available when requested by the clients.

REFERENCES

[1] D. Wu et al., “Streaming Video Over the Internet: Approaches and Directions,” IEEE Trans. Circuits and Systems for Video Technology, Mar. 2001, pp. 282-300.

[2] I. Elsen, F. Hartung, U. Horn, M. Kampmann, and L. Peters, „Streaming Technology in 3G Mobile Communication Systems“, IEEE Computer, pp. 46-52, September 2001.

[3] H. Schulzrinne, A. Rao, R. Lanphier, “Real Time Streaming Protocol (RTSP)”, RFC 2326, http://tools.ietf.org/html/rfc2326.

[4] M. Handley and V. Jacobson, and C. Perkins, "SDP: Session Description Protocol", IETF RFC 4566, April 1998.

[5] H. Schulzrinne, et al., "RTP: A Transport Protocol for Real-Time Applications", IETF RFC 3550, July 2003.

[6] 3GPP TS 26.234, Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs.

[7] 3GPP TS 26.244, Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GP).

[8] F. Gabin, M. Kampmann, T. Lohmar, C. Priddle, “3GPP Mobile Multimedia Streaming Standards”, IEEE Signal Processing Magazine, vol. 24, issue 6, pp. 134-138, November 2010.

[9] P. Fröjdh, U. Horn, M. Kampmann, A. Nohlgren, M. Westerlund, ”Adaptive Streaming within the 3GPP Packet-Switched Streaming Service”, IEEE Network, Volume 20, No. 2, pp. 34-40, March 2006.

[10] R. Fielding et al., “Hypertext Transfer Protocol -- HTTP/1.1”, IETF RFC 2616, June 1999.

[11] Open IPTV Forum, Release 2 Specification, HTTP Adaptive Streaming, V2.0, September 2010, http://www.openiptvforum.org/specifications.html.

[12] 3GPP TS 26.247, Transparent end-to-end Packet-switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH).

[13] International Multimedia Telecommunications Consortium, http://www.imtc.org/.

[14] Streaming Media Enabler on Ericsson Labs: https://labs.ericsson.com/apis/streaming-media/documentation#Adaptive_HTTP_Streaming_DASH