Pattern template manifest for live video streaming

Yongjun Wu
Amazon Prime Video
Seattle, USA
yongjuw@yongjuw.com

Kyle Koceski
Amazon Prime Video
Seattle, USA
kyleko@amazon.com

Mairo Pedrini
Amazon Prime Video
Seattle, USA
pedrinim@amazon.com

Sally Cheng
Amazon Prime Video
Seattle, USA
sallych@amazon.com

Parminder Singh
Amazon Prime Video
Seattle, USA
parmindr@amazon.com

***Abstract*— In live video streaming, the size of manifest grows linearly as the overall time duration of manifest increases in many scenarios. Such a behavior exists across streaming technologies, e.g. Dynamic Adaptive Streaming through HTTP (DASH), HTTP Live Streaming (HLS) and Microsoft Smooth Streaming (MSS). It introduces significant overhead for manifest generation on cloud services, download latency and network traffic, manifest parsing and manifest storage on both cloud services and more important on devices with limited memory and CPU power, especially in the scenario of live video streaming where manifests have to be refreshed every several seconds. In this paper, we analyze what the problem is, where it comes from according to the basics in audio and video compression properties, and why it exists across the industry of live video streaming. We also look into a couple of case studies from some content providers and popular streaming services. Then we propose the novel pattern template, which can optimize the size of manifest, completely eliminate the factor of linear growth in time about the size of manifest, and make it stay constant in the scenario of continuous streaming without discontinuities. On service side, it saves computation for manifest generation, storage and parsing and optimizes the network traffic. On device side, it optimizes the download and manifest refresh latency, manifest parsing and storage in memory. The pattern template is applicable to any adaptive bit rate video streaming techniques and protocols at network application layer, such as HLS and Smooth Streaming, though this paper focuses on DASH streaming technology. Recently pattern template manifest has been adopted in DASH standard 6th edition in 2024.**

Keywords*— *Pattern template, adaptive bit rate streaming, DASH, HLS, Smooth Streaming, segmentation, segment duration, manifest

I. Introduction

Video streaming over the internet started back in the 1990s with timely delivery and consumption of large amounts of data being the main challenge. With the increase of internet bandwidth and the tremendous growth of the World Wide Web, now video contents can be delivered efficiently in large segments using HTTP. HTTP streaming has a few main benefits [18]. First, the internet infrastructure has evolved to support HTTP efficiently. Content Delivery Networks (CDN) provide localized edge caches and reduce long-haul traffic to devices. HTTP is firewall friendly and almost all firewalls today are configured to support HTTP outgoing connections. HTTP server technology is a commodity and therefore supporting HTTP streaming for millions of users can scale up easily. Second, with HTTP streaming the client manages the streaming without the need of maintaining a session state on the server. Provisioning a large number of streaming clients doesn't impose any additional costs on server resources, beyond the standard web use of HTTP, and can be managed by a CDN using standard HTTP techniques. For all the reasons, HTTP streaming has become a popular approach in commercial deployment with a few variants on the market today, such as Apple HLS [13], MSS [14], Adobe HDS [16], and MPEG DASH [4]. Different designs use different manifests and segment formats, to receive the contents from servers.

On the other hand, internet video streaming had significant growth in the past several years. Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper [17] showed that globally IP video traffic would be 82% of all IP traffic (both business and consumer) by 2022, up from 75% in 2017. Global IP video traffic grew four-fold from 2017 to 2022, a Compound Annual Growth Rate (CAGR) of 29%. Internet video traffic grew fourfold from 2017 to 2022, a CAGR of 33%. Live internet video accounted for 17% of Internet video traffic by 2022. Live video grew 15-fold from 2017 to 2022.

Popular live streaming content providers on the market, such as BBC, Amazon Prime Video, NBC, Hulu, Facebook, HBO Max and Amazon Twitch, deploy HLS, DASH, MSS, Adobe HDS or others for video streaming. Physical partition with separate small segment files is suitable for live streaming where fragments in the future are not available yet at current time. HTTP requests are simply file based pointing to one individual and independent segment file. In live streaming, there is one interesting and common challenge about the size of manifest, which might grow linearly as time duration increases. The same challenge exists in HLS, DASH, MSS and others. It introduces significant overhead for manifest generation, download and network traffic, manifest parsing and manifest storage on both cloud services and more important on devices with limited memory and CPU power, especially when the total duration of the full manifest is long.

In this paper, we first present what the problem is in section II. In section III, we analyze the background about why such a behavior exists. In section IV, we dive deep with case studies and some mathematic analysis. In section V, we propose the novel pattern template to make manifest size stay constant without the linear growth in time and present some experimental results. In the end, in section VI we will talk about extensions, limitations and next steps.

II. What is the problem?

Let's use DASH manifest as an example starting from here, though manifests in other streaming technologies, such as HLS and MSS, have similar behaviors. The observation is that the size of DASH manifest in live video streaming grows

linearly as time duration increases. As shown in the full sample manifests of [1, 2], the audio SegmentTimeline representation grows linearly as the total duration grows and the number of audio fragments increases, as shown in Figure 1 below. In the full sample manifests of [1, 2], the audio and video segment representations are traditional, using $Number$ in DASH specification [4].

```
<S t="178577070976" d="95232"/>
<S t="178577166208" d="96256" r="2"/>
<S t="178577454976" d="95232"/>
<S t="178577550208" d="96256" r="2"/>
……
```

Fig. 1. Linear growth of audio SegmentTimeline against time duration

Such a behavior in audio SegmentTimeline is common across live video streaming technology and industry. For example, Unified package [5], AWS Elemental MediaPackage [6] and Hulu live streaming [7] all have the same behavior. The linear growth of DASH manifest size against the total time duration impacts live streaming more than video on demand, since live manifest has to be updated periodically every X seconds, where X could be 2 or 4 seconds. The behavior compromises a few scenarios in live streaming, such as start-over/scrub back in live streaming longer than N hours, where impact will be significant as long as N is larger than half an hour, Just After Broadcast (JAB) video assets in long duration, and live manifest with M minutes live window. The impacts are in a few dimensions, all leading to compromised customer experiences: manifest generation on cloud services, manifest parsing on cloud services and devices, manifest storage on cloud services and devices, and manifest download through internet pipe from cloud services to devices.

III. WHY THE BEHAVIOR EXISTS?

The fundamental reason for the presence of such a behavior is that segmentation algorithm always aligns audio and video segments in the best effort, as shown in Figure 2 below and also in the sample DASH manifest of [1]. Figure 2 has video stream in H.264/AVC [10] at 30 fps with 2-second segments, and audio stream in AAC-LC [9] at 48 KHz sampling frequency with approximately 2-second segments. Audio and video segments perfectly aligned every 8 seconds. The misalignment between corresponding audio and video segments are minimum in a pattern of [0, 5.3, 10.7, 16] ms.

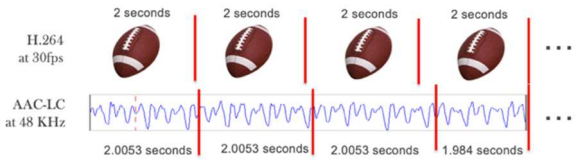

Fig. 2. 8 seconds for perfect audio and video alignment

Why the alignment between audio and video fragments are necessary? There are multiple popular reasons behind the scene.

- Trimming a certain portion of audio and video streams from the whole content will be easy. It can directly grab audio and video segments starting from the same segment numbers and ending at the same segment numbers.
- Some Media Source Extensions/Encrypted Media Extensions (MSE/EME) based players can only play the full audio and video fragments, without dropping any audio or video frames. It is a good practice to require misalignment of audio and video fragments under the audio and video synchronization threshold [-45, 125] ms [8] since those players can play the full audio and video segments assuming nearly perfect alignment in the beginning of corresponding audio and video segments.
- Playback seek will be easy to implement, with minimum audio and video drift and without dropping any audio or video frames.

Moreover, there are additional proprietary requirements from specific media services.

- Some media services require the same number of segments in audio and video streams for easy management of media timeline.
- Some media services also require audio and video to align perfectly at a certain pattern and cycle due to its proprietary service for live streaming.

All the above constraints limit ourselves from using constant audio segment duration in the same way as video segment duration, which will introduce huge drifts between corresponding audio and video segments and might never align audio and video segments at any time locations. For example, if video segments are at 2-second constant duration and audio segments are at 1.984-second constant duration, the audio and video stream misalignment will increase as 16, 32, 48, 64, 80, 96, 112, 128, 144, 160 ms, …, at the end of corresponding audio and video fragments.

IV. WHERE DOES THE BEHAVIOR COME FROM?

In this section, we will look into where such a behavior come from about audio and video segment duration in the manifest. We have two typical case studies to reveal the underneath fundamentals. The first case study is with integer video frame rate 30 fps and the most popular audio sampling frequency 48 KHz for AAC-LC audio. The second case study is with non-integer video frame rate of 29.97=30000/1001 fps and most popular audio sampling frequency 48 KHz for AAC-LC audio. The case studies are generic, applicable and extensible to other integer and non-integer video frames, and other audio codec formats, such as Dolby Digital Plus [11] and Dolby Atmos [12], and other audio sampling frequencies, such as 32 KHz.

A. *Case study 1*

When video stream has integer video frame rate 30 fps, and audio stream has sampling frequency 48 KHz for AAC-LC, ideally we will have the following equation in order to align audio and video segments perfectly.

$$X \times \frac{1}{30} = Y \times \frac{1024}{48000} \qquad (1)$$

$$\frac{X}{Y} = \frac{16}{25} \quad (2)$$

Equation (1) means what the number of video frames in duration 1/30 seconds, X, should be and what the number of audio frames in duration 1024/48000 seconds [9], Y, should be, in order for audio segments in Y frames and video segments in X frames to align perfectly. Equation (2) indicates the minimum number of X is equal to 16 and the minimum number of Y is equal to 25, and audio and video segments can align perfectly at 0.53 second or the integer multiplication of 0.53 second. Video segment duration 0.53 second or the integer multiplication of 0.53 second requires to set video encoding configuration of Instantaneous Decoder Refresh (IDR) interval [10, 15] and video segment duration carefully with audio segmentation matching video segment duration. In the real world, in general we set video segment duration in 2 seconds for integer video frame rate. Then we have two slightly different equations below.

$$F \times 2 = Y' \times \frac{1024}{48000} \quad (3)$$

$$\frac{F}{Y'} = \frac{4}{375} \quad (4)$$

Equation (3) means what the number of video segments in duration 2 seconds, F, should be and what the number of audio frames in duration 1024/48000 seconds [9], Y', should be, in order for audio segments in Y' frames and F video segments in 2-second duration to align perfectly. Equation (4) indicates the minimum number of F is equal to 4 and the minimum number of Y' is equal to 375, and audio and video segments can align perfectly at every 8 seconds or the integer multiplication of 8 seconds. 8 seconds can consist of 4 2-second video segments in general in live streaming, and 4 audio fragments in (94, 94, 94, 93) audio frames, which have the time duration pattern of (2.0053,2.0053,2.0053, 1.984) seconds. This is where the typical pattern and cycle in Elemental MediaPackage [1] and Unified package come from.

B. *Case study 2*

When video stream has non-integer video frame rate, $29.97 = 30000/1001$ fps, and audio stream has sampling frequency 48 KHz, similarly we will have the following equations in order to align audio and video segments perfectly in the ideal world.

$$A \times \frac{1001}{30000} = B \times \frac{1024}{48000} \quad (5)$$

$$\frac{A}{B} = \frac{640}{1001} = \frac{1920}{3003} \quad (6)$$

Equation (5) means what the number of video frames in duration (1001/30000) seconds should be and what the number of audio frames in duration (1024/48000) seconds [9], B should be, in order for audio segments in A frames and video segments in B frames to align perfectly. Equation (6) indicates the minimum number of A is equal to 640 and the minimum number of B is equal to 1001, and audio and video segments can align perfectly at 21.3547 seconds or the integer multiplication of 21.3547 seconds. For example, the alignment of audio and video segments can happen at 64.064=3×21.3547 seconds. 64.064 seconds have 16 4.004-second video fragments at 29.97 fps, and 5 3.989-second and 11 4.011-second audio segments. That is the pattern and cycle in sample manifest from Hulu live streaming [2].

C. *General observations and conclusions*

From the two typical and generic case studies, we can draw some conclusions.

- Audio and video segments can be aligned perfectly in integer frame rate and most popular sampling frequency of 48 KHz for AAC-LC with careful configurations.
- General practice of 2-second video segment duration in video streaming will introduce a typical pattern and cycle of audio fragment duration even with integer video frame rate
- We have to have a pattern and cycle of audio segment duration with non-integer frame rate and AAC-LC sampling frequency of 48 KHz.

Similar observations and conclusions can be extended to Dolby Digital Plus [11], Dolby Atmos [12], and other encoding modes of AAC [9], such as HE-AAC v1 and HE-AAC v2, and/or other audio sampling frequencies, such as 32 KHz, though other audio sampling frequencies are not popular and used by content providers in video streaming. Whenever there is a pattern of audio segment duration, it is inevitable to have DASH manifest linearly growing when the total time duration of DASH manifest increases given the existing syntaxes available in DASH specification.

V. PATTERN TEMPLATE MANIFEST AND RESULTS

DASH specification [4] already has the run-length compression of segment duration defined in the syntax @r. Quote the definitions from DASH specification [4]: "@r specifies the repeat count of the number of following contiguous Segments with the same duration expressed by the value of @d. This value is zero-based (e.g. a value of three means four Segments in the contiguous series). A negative value of the @r attribute of the S element indicates that the duration indicated in @d attribute repeats until the start of the next S element, the end of the Period or until the next MPD update".

```
<SegmentTimeline>
   <S t="111610668" d="60" r="4244"/>
</SegmentTimeline>
```

Fig. 3. Sample run-length compression of video segment SegmentTimeline

It's the first-order compression of segment duration in a DASH manifest. It only works for repeated segment duration, not for segment duration in a pattern or cycle. It's natural to have the definition of the new syntax "Pattern" for DASH specification as follows: Element **Pattern** to allow for the grouping of a set of **S** elements into a pattern. **Pattern** element may exist in conjunction with **S** elements at the SegmentTimeline level and contains child **S** elements which represents a repeating pattern of multiple segments.

```
<SegmentTimeline>
   <Pattern r="1" t="178577070976" >
      <S d="95232"/>
      <S d="96256" r="2"/>
```

```
    </Pattern>
    <S t="178577838976" d="95232"/>
  </SegmentTimeline>
```

Fig. 4. Sample pattern template compression of audio segment SegmentTimeline. Please see [3] for the full details.

Figure 4 shows the example of pattern template compression of audio SegmentTimeline. Obviously, pattern template has the following properties and benefits.

- It is second-order compression of segment duration in a pattern.
- It is applicable to JAB, Start-Over, live manifest and more.
- It makes (live) manifest size constant, not dependent on total time duration of N hours when there is no discontinuity in media timeline.
- It optimizes manifest generation, parsing and storage on services and devices, and (refresh) download through internet pipes, and reduces overall manifest data overhead significantly.

As shown in [1, 3], the manifest size reduction is obvious. A DASH manifest in the duration of 2.3 hours reduces its size from 121 KB to 24 KB, more than 5 times, with pattern template SegmentTimeline, compared to the original SegmentTimeline without pattern template. If the time duration is longer, e.g. 5 hours, the reduction will be more than 10 times. On the other hand, if there are multiple (M) audio codec formats and/or if there are multiple (N) audio tracks for different languages, the reduction will be M times, N times, or $M \times N$ times more, fully dependent on the number of audio streams. It is obvious that under the same hardware/CPU configurations the time spent on manifest generation and parsing on both services and devices will be reduced by $M \times N$ times, under the same network condition the time spent on download will be reduced by $M \times N$ times, and the memory used for manifest storage on both services and device will be reduced by $M \times N$ times. On service side, the CPU consumption and memory storage could be mitigated by using some cloud instance at a higher performance. It has direct impacts on customer video experiences about power consumption, time to start playback, and response time under high CPU and memory usage, especially in the scenario of periodical manifest refresh of a big manifest at a long total duration.

We can have some further optimizations that different **Patterns** are stored at a higher level beyond the Representation in DASH manifest with an ID, such as MPD level, and re-used at lower level through the ID reference. The reuse of patterns can further optimize the scenarios of multiple audio codec formats and/or multiple audio tracks for different languages, and live streams with discontinuities, due to either ad insertion or streaming failures etc. Figure 5 shows one typical example that **Patterns** are stored at a higher level but SegmentTimeline simply refers to a Pattern ID without repeating the details.

```
<Patterns>
    <Pattern id="1">
        <S d="95232"/>
        <S d="96256" r="2"/>
    </Pattern>
</Patterns>
<SegmentTimeline>
    <Pattern r="1" t="178577070976" id="1" />
    <S t="178577838976" d="95232"/>
</SegmentTimeline>
```

Fig. 5. Patterns defined at a higher level but reused at a lower level.

VI. Extensions and Conclusions

Pattern template is also applicable to HLS, Smooth Streaming and other streaming technologies beyond DASH, since the pattern comes from audio and video compression properties, i.e. video frame rate and duration, and audio frame duration decided by sampling frequency and the number of audio point samples in one audio frame in a certain audio coded format. Since pattern template is not part of HLS specification yet [13], we need some manifest proxy to unpack the pattern into flat HLS manifest before sending to Apple player. But we can still achieve most of optimizations and savings, if cloud services generate and deliver pattern template in HLS manifest to the HLS manifest proxy. If video streaming services have the end-to-end controls over both cloud services for manifest generation, and players for manifest download and parsing, pattern template can be deployed, even if it is not part of HLS or Smooth Streaming specification yet. The limitation of pattern template is that it compresses well the audio media timeline within a period of contiguous audio streams, not across periods, since period boundaries interrupt the audio media timelines. Hence it works best when the long-duration manifest is contiguous, or only has limited discontinuities. When there are lots of discontinuities, such as many periods in DASH or discontinuities in HLS, we might need additional optimizations using other technologies [21], such as Patch Manifest [4] and Compacted Manifest [19], co-existing together with pattern template manifest.

Recently pattern template manifest has been adopted by MPEG-DASH 6th edition [20] in 2024. We will drive wide adoption and deployment of pattern template manifest, benefiting everyone in live streaming industry and optimizaing data overhead across the video streaming ecosystem, and at the same time drive the adoption in HLS, together with robust designs in failure scenarios.

References

[1] https://github.com/yongjuw-git/manifests/blob/main/Elemental-2.3-hours.mpd

[2] https://github.com/yongjuw-git/manifests/blob/main/Hulu.mpd

[3] https://github.com/yongjuw-git/manifests/blob/main/Pattern.mpd

[4] ISO/IEC 23009-1 Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.

[5] https://unified-streaming.com/products/unified-packager

[6] https://aws.amazon.com/mediapackage/

[7] https://www.hulu.com/

[8] https://en.wikipedia.org/wiki/Audio-to-video_synchronization

[9] ISO/IEC 13818-7 Information technology — Generic coding of moving pictures and associated audio information — Part 7: Advanced Audio Coding (AAC)

[10] ITU-T H.264 Advanced video coding for generic audiovisual services

[11] ATSC Standard:Digital Audio Compression (AC-3, E-AC-3)

[12] https://www.dolby.com/us/en/technologies/dolby-atmos/dolby-atmos-specifications.pdf

[13] https://tools.ietf.org/html/draft-pantos-http-live-streaming-23

[14] https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SSTR/%5bMS-SSTR%5d.pdf

[15] ISO/IEC 23008-2 Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding

[16] https://www.adobe.com/devnet/hds.html

[17] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[18] I. Sodagar, The MPEG-DASH Standard for Multimedia Streaming Over the Internet, Page: 62 - 67, IEEE Multimedia, Vol. 18 , Issue 4 , April 2011.

[19] Compacted DASH manifests - AWS Elemental MediaPackage (amazon.com)Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[20] https://github.com/Dash-Industry-Forum/Dash-Industry-Forum.github.io/files/14887658/SpecialPresentation-6thEdition.pdf

[21] Y. Wu and K. Koceski, The analysis of DASH manifest optimizations, Pages: 28–32, GMSys '23: Proceedings of the First International Workshop on Green Multimedia Systems, June 2023.