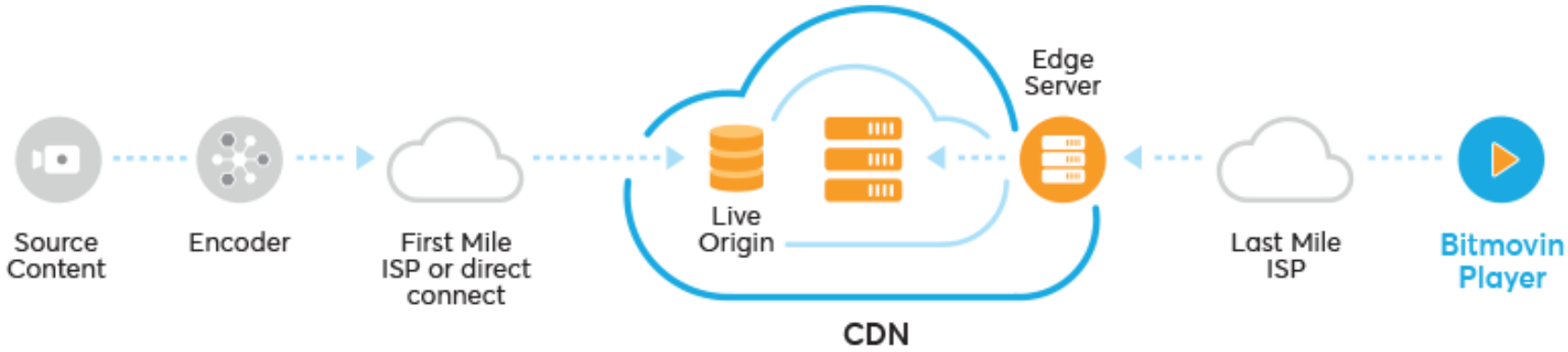
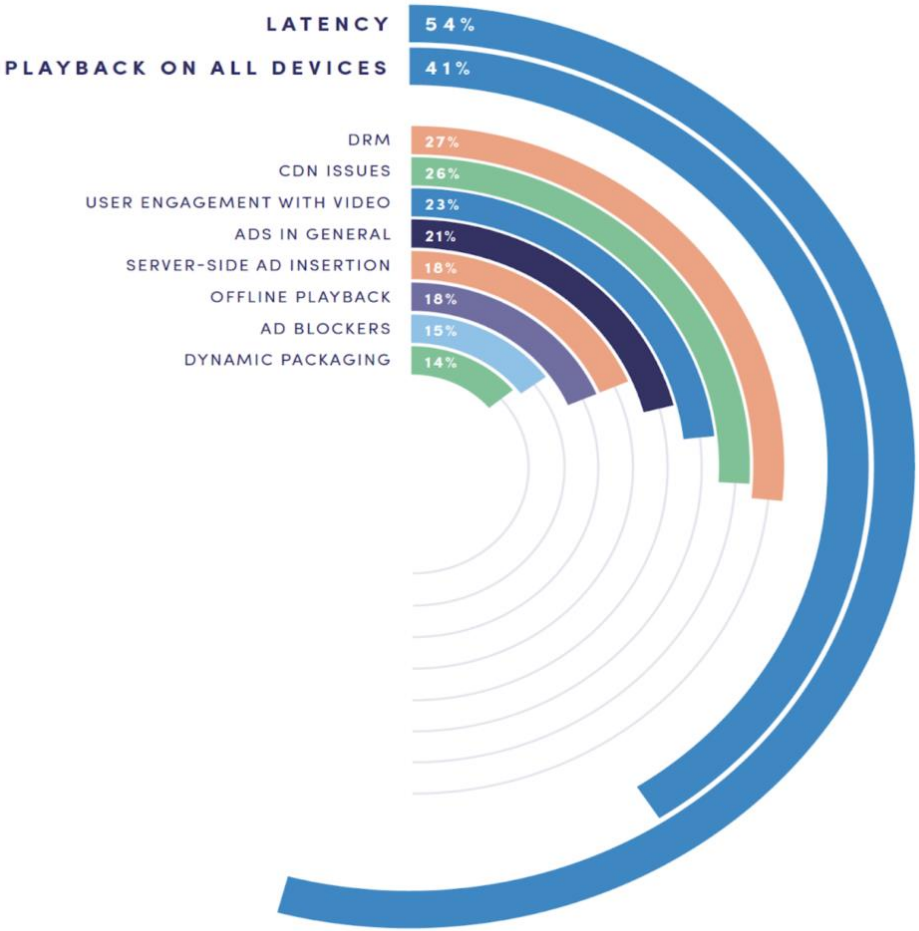
What is Live Low Latency?

Low Latency in live streaming is the time delay between an event's content being captured at one end of the media delivery chain and played out to a user at the other end. Consider a goal scored at a football game: Live latency is the delay in time between the moment a goal is scored and captured by a camera until the moment that a viewer sees the goal on their own device. There are a few different terms that effectively define the same experience: end-to-end latency, hand-waving latency, or glass-to-glass latency.



End-to-end video encoding workflow (where latency matters)

In our most recent developer report, low latency was identified as one of the biggest challenges for the media industry. This blog series will take an in-depth look into why that's the case, welcome to our Live Latency Deep Dive series!



Why care about Low Latency?

Most use cases where live latency is crucial can be categorized into the following:

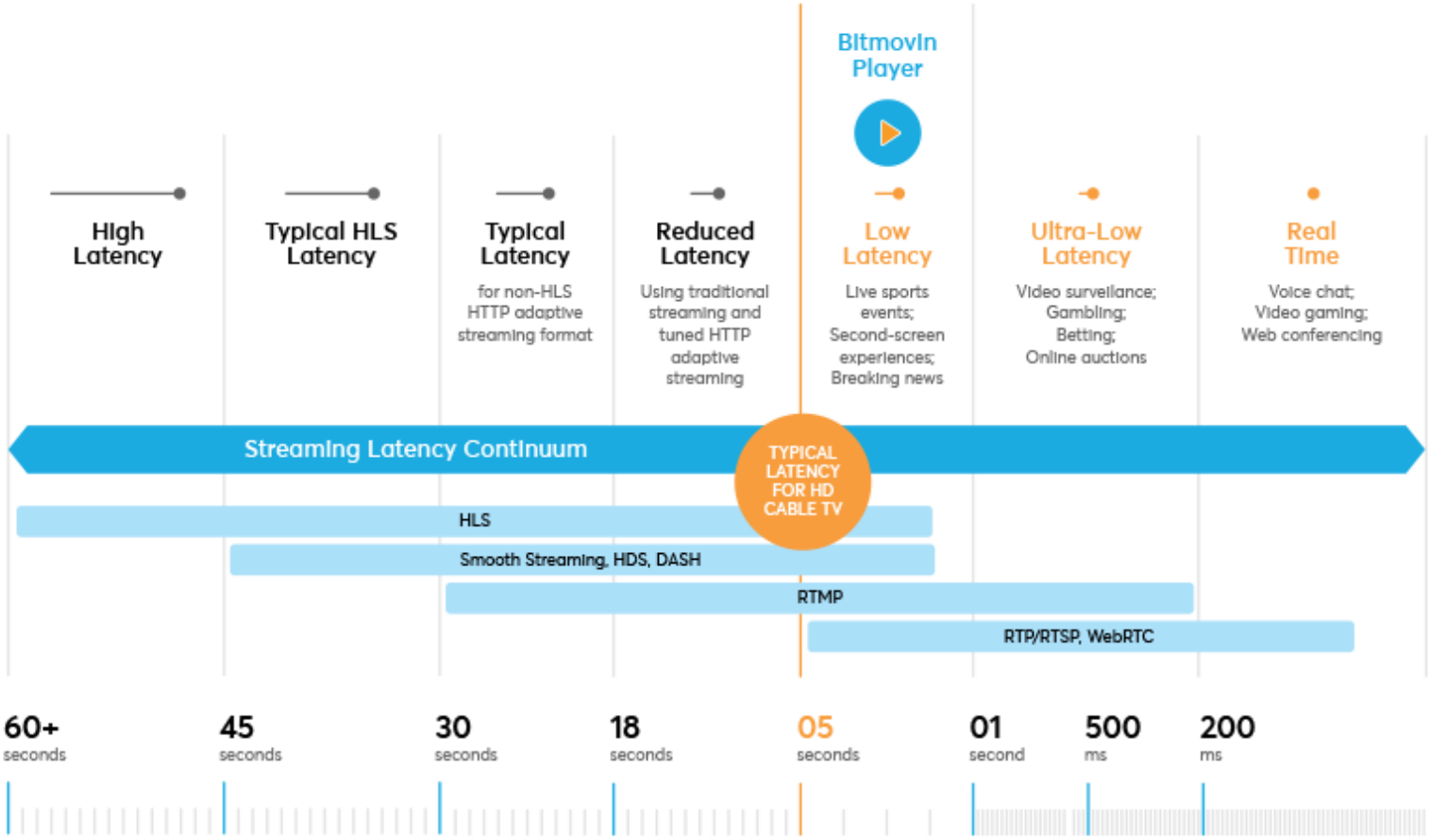
Live content delivered across multiple distribution channels

high live latency in comparison to traditional linear broadcast delivery via satellite, terrestrial or cable services. Over-the-top (OTT) delivery methods like MPEG-DASH and Apple HLS have become the defacto standard for delivering video to audiences using mobile devices such as smartphones, tablets, laptops, and Smart TVs. Live network content, like sports or news, drive the need for low live latency as these networks attempt to deliver content simultaneously over various distribution means (e.g. OTT vs Cable).

Picture a scenario where you are streaming your favourite football team playing in the global final, your neighbour and equal fan (with incredibly thin walls) has traditional linear cable. It's the final moments of the game, but you hear the neighbour cursing loudly, despite the fact that there is well over 1 minute left in the game. The thrill is spoiled and you know your team certainly lost. The need for faster live latency becomes clear, the difference between broadcast and streaming is unacceptable in today's digital world. But a lot of factors affect how quickly content will appear on a viewer's screen. Aside from infrastructural issues (like not being optimized for low latency), modern streaming methods may suffer latency delays from additional factors like social media feeds, push notifications, and second-screen experiences running in parallel to the live event.

Interactive live content

Whenever audience interaction is involved, live latency should be as low as possible to ensure a good quality of experience (QoE). Such use cases include webinars, auctions, user-generated content where the broadcaster interacts with the audience (e.g. Twitch, Periscope, Facebook Live, etc.) and more. Latency is often measured on a spectrum, where high latency is the least sought after delay, and *Real-Time* is the most sought after. See the Latency Spectrum below (including the latency types, delay time, and streaming formats):



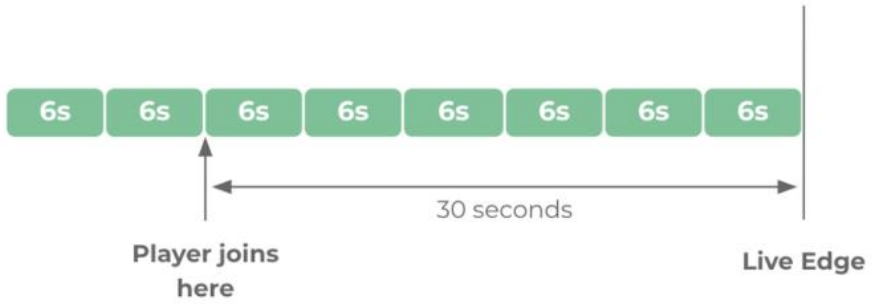
The latency spectrum shows that unoptimized OTT delivery accounts for around 30+ seconds of delay while cable broadcast TV clocks in at around 5 seconds – give or take. Furthermore, sub-second latencies may not be achievable with OTT methods and require other protocols like WebRTC.

Where does live latency come from?

First, a slightly more technical definition of live latency: It's the time difference between a video frame being captured and the moment it's presented to the playback client. In other words, it's the time that a video frame spends in the media processing and delivery chain. Every component in the chain introduces a certain amount of latency and eventually accumulates to what is considered live latency.

Let's have a look at the main sources of live latency:

Buffering ahead for playback stability at the player-level



Live stream timeline

A video player will aim to maintain a pre-defined amount of buffered data ahead of its playback position. The standard value is about 30 seconds of buffer loaded ahead at all times during playback. One of the reasons behind this is the cause is that if network bandwidth drops during playback there would still be 30 seconds of data to be played out without interruption? During this time the player can react to new bandwidth conditions

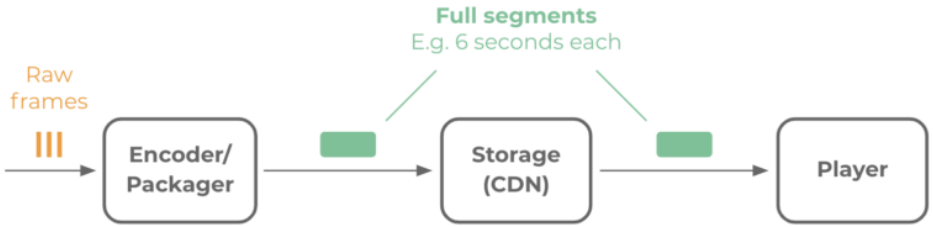
appropriately, thereby buying the player some time to adapt. Buffer time also typically influences the bitrate adaptation decisions as low buffer levels may imply more aggressive downwards adaptations.

However, when aiming for 30 seconds of buffer with a live stream, the player must stay at least 30 seconds behind the live edge (the most recent point) of the stream with its playback position; this would result in a live latency of 30 seconds. Conversely, this means that aiming for a low latency would require being even closer to the live edge and implies having a minimum buffer. If we aim for 5 seconds of latency, the player would have 5 seconds of buffer at most. Thus, the difficult decision of trading off between latency and playback stability must be made.

Segments are produced, transferred and consumed in their entirety

Live streams are encoded in real-time. This means that if a segment duration is 6 seconds it will take the encoder 6 seconds to produce one full segment. Additionally, if fragmented MP4 is used as the container format, encoders can only write a segment to the desired storage once it's encoded completely, i.e. 6 seconds after starting the encode of the segment. So once a segment is transferred to the storage its oldest frame is already 6 seconds old. On the other side of the delivery chain, the player can only decode an fMP4 segment in its entirety and therefore needs to download a segment fully before it can process it. Network transfers: like uploading a video to a CDN origin server, transferring the content within the CDN, and downloading from the CDN edge server to the client can add to the overall latency to a lower degree.

In summary, the fact that segments are only processed and transferred in their entirety results in latency being correlated directly to segment duration.



Data Segments in the Encoding Workflow

What can we do?

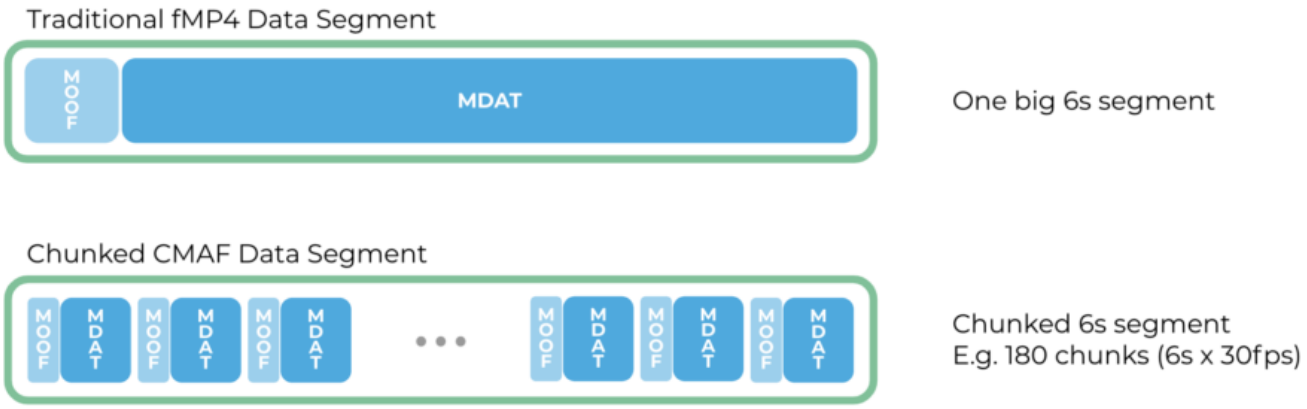
Naive approach: Short segments

As latency is correlated to segment duration, a simple way to decrease latency would be to use short segments, e.g. 1-second duration. However, this comes with negative side effects such as:

- **Video coding efficiency suffers**: The requirement of each video segment starting with a key frame implies having small groups of pictures (GOPs). This in turn, causes the efficiency of differential/predictive coding to suffer. With short segments, you'd have to spend more bits if you're aiming for the same perceptual quality as longer segments with the same content.
- **More network requests** and everything negative associated with them, e.g. time to first byte (TTFB) wasted on every request.
- **Increased number of segments** may decrease CDN caching efficiency.
- **Buffer at the player grows** in a jumpy fashion which increases the risk of playback stalls due to rebuffering.

Chunked encoding and transfer

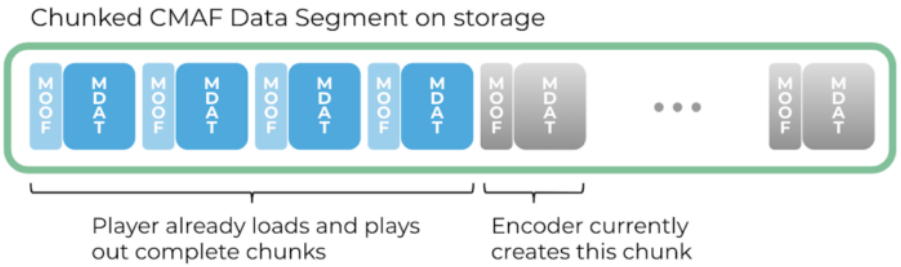
To solve the problem of segments being produced and consumed only in their entirety, we can make use of the chunked encoding scheme specified in the MPEG-CMAF (Common Media Application Format) standard. CMAF defines a container format based on the ISO Base Media File Format (ISO BMFF), similar to the MP4 container format, which is already widely supported by browsers and end devices. Within its chunked encoding feature, CMAF introduces the notion of CMAF chunks. Compared to an "ordinary" fMP4 segment that has its media payload in a single big mdat box, chunked CMAF allows segments to consist of a sequence of CMAF chunks (moof+mdat tuples). In extreme cases, every frame can be put into its own CMAF chunk. This enables the encoder to produce and the player's decoder to consume segments in a chunk-by-chunk fashion instead of limiting use to entire segment consumption. Admittedly, the MPEG-TS container format offers similar properties as chunked CMAF, but it's fading as a format for OTT due to the lack of native device and platform support that fMP4 and CMAF provide.



6s fMP4 segment compared to chunked CMAF

Chunked encoding on its own does not help us decrease the latency but is a key ingredient. To capitalize on chunked encodes, we need to combine the process with HTTP 1.1 chunked transfer encoding (CTE). CTE is a feature of HTTP that allows resource transfers where size is unknown at the time of transfer. It does so by transferring resources chunk-wise and signaling the end of a resource with a chunk of length 0. We can utilize CTE at the encoder to write CMAF chunks to the storage as soon as they are being produced without waiting for the encode of the full

segment to finish. This enables the player to request (also using CTE) available CMAF chunks of a segment that is still being encoded and forward them as fast as possible to the decoder for playout. Therefore allowing playback as soon as the first CMAF chunk is received.



Implications of low latency chunked delivery

… besides enabling low latency:

- **Smoother and less jumpy client buffer levels** from the constant flow of CMAF chunks received. Thus lowering the risk of buffer underruns and improves playback stability.
- **Faster stream startup** (*time to first frame*) and seeking at the client due to being able to decode and playout segments partially during their download.
- **Higher overhead in segment file size** compared to non-chunked segments as a result of the additional metadata (moof boxes, mdat headers) introduced with chunked encodes.

- **Low buffer levels** at the client impact playback stability. A low live latency implies the client is playing close to the live edge and has a low buffer level. Therefore the longest achievable buffer level is limited by the current live latency. It's a QoE tradeoff: low latency vs. playback stability.
- **Bandwidth estimation for adaptive streaming at the client is hard.** When loading a segment at the bleeding live edge, the download rate will be limited by the source/encoder. As content is produced in real-time it takes, for example, 6 seconds to encode a 6-second long segment. So the download rate/time for segments is no longer limited by networks but by encoders. This causes a problem in bandwidth estimation methods that are currently commonplace in the industry and based on the download duration. The standard formula to calculate bandwidth estimation is:

estimatedBW = segmentSize / downloadDuration

E.g.: estimatedBW = 1MB / 2s = 4mbit

As download duration roughly equals the segment duration when loading at the bleeding live edge using CTE, it can no longer be used to estimate client bandwidth. Bandwidth estimation is a crucial part of any adaptive streaming player and the lack of estimated bandwidth must be addressed. Research for better ways to estimate bandwidth in chunked low-latency delivery scenarios is ongoing in academia and throughout the streaming industry, e.g. ACTE.

Did you enjoy this post? Want to learn more? Check out Part two of the Low Latency series: Video Tech Deep-Dive: Live Low Latency Streaming Part 2

This blog post is continuation of an ongoing blog and webinar technical deep series. You can find the first blog post here and the associated webinar recording here. The first post covered the fundamentals of live low latency and defined chunked delivery methods with CMAF.

This blog post expands on chunked CMAF delivery by explaining it's application with MPEG-DASH to achieve low latency. We'll lay some foundations and cover the basic approaches behind low-latency DASH, then look into what future developments are expected as low-latency streaming is a heavily researched subject and is quickly becoming a media industry standard.

Basics of MPEG-DASH Live Streaming

Before diving into how Low Latency Streaming works in MPEG-DASH we first need to understand some basic stream mechanics of DASH live streams, most importantly, the concept of segment availability.

The DASH Media Presentation Description (MPD) is an XML document containing essential metadata of a DASH stream. Among many other things, it describes which segments a stream consists of and how a playback client can obtain them. The main difference between on-demand and live stream segments within DASH is that all segments of the stream are available at all times for on-demand; whereas the segments are produced continuously one after another as time progresses for live-streams. Every time a new segment is produced, its availability is signalled to playback clients through the MPD. It is important to note that a segment is only made available once it is fully encoded and written to the origin.

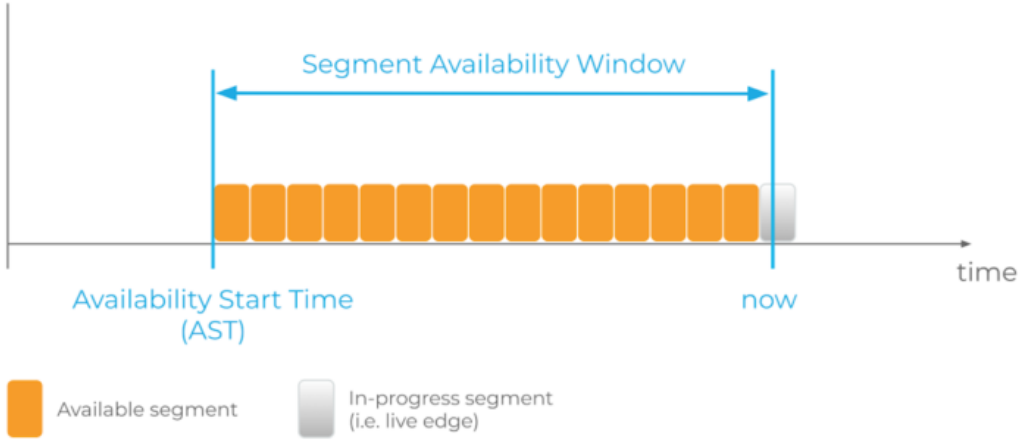


Fig. 1 Live stream with template-based addressing scheme (simplified)

The MPD would specify the start of the stream availability (i.e. the Availability Start Time) and a constant segment duration, e.g. 2 seconds. Using these values the player can calculate how many segments are currently in the availability window and also their individual availability start time. For example, the segment availability start time for the second segment would be AST + segment duration * 2.

Low Latency Streaming with MPEG-DASH

In the first part of this blog post series, we described how chunked encoding and transfer enables partial loads and consumption of segments that are still in the process of being encoded. To make a player aware of this action, the segment availability in the MPD is adjusted to signal an earlier availability, i.e. when the first chunk is complete. This is done using the availabilityTimeOffset in the MPD. As a result, the player will not wait for a segment to be fully available and will load and consume it earlier.

Consider the example of Fig.1 with a segment duration of 2 seconds and a chunk duration of 0.033 seconds (i.e. one video frame duration with 29.97 fps). To signal the segment availability once the first chunk is completed we would set the availabilityTimeOffset to 1.967 seconds (segment_duration – chunk_duration). This would signal the greyed-out segment in Fig. 1 to become partially available.

The below MPD represents this example:

```
<?xml version="1.0" encoding="utf-8"?>
<MPD
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011"
  xmlns:xlink="http://www.w3.org/1999/xlink"
 xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-
DASH_schema_files/DASH-MPD.xsd"
  profiles="urn:mpeg:dash:profile:isoff-live:2011"
  type="dynamic"
  minimumUpdatePeriod="PT500S"
  suggestedPresentationDelay="PT2S"
  availabilityStartTime="2019-08-20T05:00:03Z"
  publishTime="2019-08-20T12:42:07Z"
  minBufferTime="PT2.0S">
  <Period start="PT0.0S">
    <AdaptationSet
      contentType="video"
      segmentAlignment="true"
      bitstreamSwitching="true"
      frameRate="30000/1001">
      <Representation
       id="0"
       mimeType="video/mp4"
       codecs="avc1.64001f"
       bandwidth="2000000"
       width="1280"
       height="720"
        <SegmentTemplate
         timescale="1000000"
```

```
        duration="2000000"
        availabilityTimeOffset="1.967"
        initialization="1566277203/init-stream$RepresentationID$.m4s"
        media="1566277203/chunk-stream_t_$RepresentationID$-$Number%05d$.m4s"
        startNumber="1">
       </SegmentTemplate>
     </Representation>
   </AdaptationSet>
  </Period>
</MPD>
```

To recap, for low-latency DASH we are mainly doing two things:

- Chunked encoding and transfer (i.e. chunked CMAF)
- Signalling early availability of in-progress segments

While the previous approach enables a basic low-latency DASH setup, there are additional considerations to be made to further optimize and stabilize streaming experience. The DASH Industry Forum is working on guidelines for low-latency DASH to be released in the next version of the DASH-IF Interoperability Points (DASH-IF IOP) – expected in early July 2020. The change request for that can be found here. The following will explain key parts of these guidelines. Please note that some features were not officially finalized and standardized at the time of this post's publication (June 2020).

Wallclock Time Mapping

For the purpose of measuring latency, a mapping between the media's presentation time and the wall-clock time is needed. This is so that for any given presentation time of the stream the corresponding wall-clock time is known. The latency for a given playback position can then be calculated by determining the corresponding wall-clock time and subtracting it from the current wall-clock time.

This mapping can be achieved by specifying a so-called *Producer Reference Time* either in the segments (i.e. inband as prft box) or in the MPD. It essentially specifies the wallclock time at which the respective segment/chunk was produced. (as seen below)

```
<ProducerReferenceTime

  id="0"

  type="encoder"
```

```
  presentationTime="538590000000"

  wallclockTime="2020-05-19T14:57:45Z">

</ProducerReferenceTime>
```

The type attribute specifies whether the reference time was set by the capturing device or the encoder. Allowing for calculation of the *End-to-End Latency* (EEL) or *Encoder-Display Latency* (EDL), respectively.

Client Time Synchronization

A precise time/clock at the playback client is necessary for calculations that involve the client's wallclock time such as segment availability calculations and latency calculations. It is recommended for the MPD to include a UTCTiming element which specifies a time source that can be used to adjust for any drift of the client clock. (as seen below)

```
<UTCTiming

  schemeIdUri="urn:mpeg:dash:utc:http-iso:2014"

  value="https://time.akamai.com/?iso"

/>
```

Low Latency Service Description

A ServiceDescription element should be used to specify the service provider's desired target latency and minimum/maximum latency boundaries in milliseconds. Furthermore, playback rate boundaries may be specified that define the allowed range for playback acceleration/deceleration by the playout client to fulfill the latency requirements.

```
<ServiceDescription id="0">

  <Latency target="3500" min="2000" max="10000" referenceId="0"/>

  <PlaybackRate min="0.9" max="1.1"/>
```

```
</ServiceDescription>
```

In most player implementations such parameters are provided externally using configurations and APIs.

Resynchronization Points

The previous post pointed out that chunked delivery decouples the achievable latency from the segment durations and enables us to choose relatively long segment durations to maintain good video encoding efficiency. In turn, this prevents fast quality adaptation of the player as quality switching can only be done on segment boundaries. In a low-latency scenario with low buffer levels, fast adaptation — especially down-switching — would be desirable to avoid buffer underruns and consequently playback interruptions.

To that end, Resync elements may be used that specify segment properties like chunk duration and chunk size. Playback clients can utilize them to locate resync point and

- Join streams mid-segment, based on latency requirements
- Switch representations mid-segment
- Resynchronize at mid-segment position after buffer underruns

The previous was a glimpse of what to expect in the near future and shows the great effort of the media industry put into kick-starting low-latency streaming with MPEG-DASH and getting it ready for production services.

Want to learn more? Check out some of the supporting documentation below:

[Tool] DASH-IF Conformance Tool

[Blog Post] Video Tech Deep-Dive: Live Low Latency Streaming Part 1

[Demo] Low Latency Streaming with Bitmovin's Player

Ultra-low latency streaming with CMAF

By Dr. Andreas Unterweger28 July 2020

CMAF facilitates backwards-compatible ultra-low latency streaming. Dr. Andreas Unterweger explains why and how this is achieved and discusses exemplary practical challenges that must be overcome when implementing CMAF-compatible pipelines that provide ultra-low latency consumer experiences.

Dr. Andreas Unterweger

Many competing formats for multimedia streaming exist today, but only few are widely used. Two of the most prominent are certainly HTTP Live Streaming (HLS) and Dynamic Adaptive Streaming over HTTP (DASH). While both deliver multimedia data in a very similar manner, they are not compatible with one another. To deliver the same source audio and video data via both formats, it is necessary to store it twice in slightly different representations.

The Common Media Application Format (CMAF) solves this issue by defining a more abstract format that enables different manifests for the same encoded data. This way, the encoded data needs to be stored only once, with one manifest for HLS and one for DASH. Some additional restrictions still apply to enable full compatibility with legacy consumer HLS or DASH players, e.g., the use of fragmented MP4 to package the data. This is especially important for ultra-low latency use cases, where end-to-end delays in the second or even sub-second range are desired.

Why is latency important?
In live broadcasts, such as sporting events and concerts, minimizing latency is critical. Imagine paying for the live stream of a soccer game and

hearing your neighbor cheer “goal” five seconds before you even see it happening on your screen. These scenarios can be avoided by trying to reduce the end-to-end latency of the live stream down to a minimum.

Latency is introduced by different components within the streaming pipeline between the broadcaster and the consumer. Some components such as the data link protocol are usually already operating very close to their physical limit, e.g., the speed of light, with sub-millisecond processing delays at the sending and the receiving end. Other components such as the video encoder come in different flavors, many of which include low-latency options to reduce the delay between the last input pixel and the first encoded byte.

One component of the pipeline, however, is sometimes overlooked despite it implicitly adding significant latency – the restrictions of the streaming format. While neither HLS nor DASH explicitly add any practically relevant delays on their own, the way in which they require the sender and the receiver (player) to communicate encoded data at the live edge can add substantial delays of several seconds or even tens of seconds, depending on the size of the data segments.

- **Read more:** Live sports innovations on track

Why is chunked transfer helpful?
Segments are the groups of frames (or audio samples) that can be fetched at once. They are listed in the manifest and are typically several seconds long. Legacy HLS implementations require that each segment be added to the playlist after all its frames are fully encoded. This adds a latency equal to the duration of the segment, i.e., typically several seconds. In addition, the player can only start processing data after it is aware of the new segment in the playlist and has fetched the segment. This adds additional latency.

Imagine paying for the live stream of a soccer game and hearing your neighbor cheer “goal” five seconds before you even see it happening on your screen.

CMAF enables segments published on the encoder side to be split into chunks, which can be as small as one single frame. Once a chunk is encoded, it can be published immediately through HTTP Chunked Transfer Encoding, where each chunk as a part of the whole segment is transmitted one after another. This allows the manifest to reference a still incomplete segment that the player can already start fetching if it also supports chunked transfer. This reduces the latency on both sides.

While several updates to HLS, such as the recent Low Latency HLS (LLHLS), enable chunked transfer, they are still mostly incompatible with DASH and not widely implemented. With CMAF compatibility, it is possible to use chunked transfer of the data with two different manifests – one for LLHLS and one for DASH, each with their respective chunk-level playlist representations. Even legacy HLS can be supported this way as players unaware of chunked transfer can be served with the segment as usual once it is complete. Moreover, they still benefit from the latency reduction on the encoder side.

Why does chunked transfer sometimes fail in practice?
Nonetheless, for all latency reductions to fully manifest in real-world use cases, every component of the streaming pipeline must support chunked transfer reliably. Experience from several practical sporting event use cases, collected while building a streaming infrastructure for Austrian solutions provider NativeWaves, has shown that even a single component that does not reliably support chunked transfer can increase the end-to-end delay by multiple seconds. One particularly surprising example are software player implementation specifics.

While many players already support chunked transfer, some of them fetch chunks one after another instead of in parallel or within one connection (via keep-alive). This may cause very small chunks to not be fetched fast enough so that the buffer constantly underflows and playback stutters. If the player implementation cannot be modified, such issues can usually only be fixed by falling back from the live edge and/or by resorting to fetching full segments instead of chunks, which increases latency.

The same is true when player implementations without keep-alive support reach the limits of the receiver operating system, such as the maximum number of concurrent TCP connections. When this number is exceeded, playback may fail with hard-to-pinpoint symptoms such as smooth low-latency playback for multiple minutes, followed by a complete halt of playback until the system-specific timeout settings allow for new connections again.

Clearly, one single component in the streaming pipeline may be sufficient to prevent ultra-low-latency streaming. While CMAF certainly enables playback of ultra-low latency streams for different existing manifest formats, even with backwards-compatibility, it will still take a significant amount of time until all components of the pipeline are ready to actually allow for smooth ultra-low latency end-to-end streaming in practice.

- **Read more:** Closing the gap between broadcast and OTT

About the author
Dr. Andreas Unterweger is a teaching professor for Media Technology at the Salzburg University of Applied Sciences, with a background in video coding, streaming and multimedia security. He is also the Lead Video Encoding Engineer at NativeWaves, a Salzburg-based company which provides synchronized multi-screen streaming for low-latency use cases.