Introduction

Over the last 15 years, live streaming services have grown from novelties and experiments into profitable businesses serving an ever-growing cohort of cord-cutters and cord-nevers. Initial streaming implementations mimicked the workflows of the broadcast world, using custom servers to deliver streams via proprietary protocols. Here at Akamai, we’ve witnessed the evolution of over-the-top (OTT) live streaming as traffic grew 22,000-fold from a 1 Gbps stream in 2001 (the first Victoria’s Secret fashion show webcast) to 23 Tbps in June 2018 for a global football tournament.

What enabled this dramatic growth in volume? One primary factor was the transition in the mid-2000s to HTTP Adaptive Streaming (HAS), which used standard HTTP servers and TCP to deliver the content, allowing content delivery networks (CDN) to leverage the full capacity of their HTTP networks to deliver streaming content instead of relying upon smaller networks of dedicated streaming servers.

Early HAS services were cautious with this new delivery medium and hence Apple, with their HTTP Live Streaming (HLS) format, recommended 10-second segments and a conservative three-segment starting buffer in the player. These settings became a de facto standard for streaming that was adopted across the industry, making 40-second latencies \(^1\) the norm and cementing the mindset for many that over-the-top (OTT) delivery would always be the laggard behind broadcast. In the broadcast world, satellite latencies are in the 3.5-12 s range and terrestrial cable between 6-12 s, depending on your location in the world. Apple has since lowered their recommended segment duration to 6 s, but their default player implementation still requires three segments of starting buffer, which continues to add latency to a live stream.

The pressure for OTT to reduce end-to-end latency to match or beat broadcast can in some part be achieved by shortening segment durations. However, ever-decreasing segment sizes force ever-shortening group-of-picture (GOP) lengths that diminish visual quality, and segment duration that is only a few multiples of the round-trip time leads to unstable delivery infrastructure in CDNs.

Is there another way to efficiently deliver lower latency at scale while preserving visual quality of a video? The answer is yes.
CMAF File Containers

Our journey to low latency at scale starts with a common file format. HLS and MPEG DASH are the two dominant HAS formats in the market today. HLS has traditionally used TS containers to hold the muxed audio and video data, while DASH preferred the ISO-Base Media File Format holding demuxed tracks. This means that content owners wanting to reach the diversity of devices must package and store two sets of files, each holding exactly the same audio and video data. These two containers also contend with each other for CDN space, making CDN delivery less efficient.

Microsoft and Apple realized these inefficiencies and instigated the development and standardization within MPEG of the Common Media Application Format (CMAF) in 2017. CMAF is a restriction upon the well-established fragmented mp4 container and is very similar to the DASH-ISO file format.

CMAF is simply a standardized container that can hold video, audio, or text data. CMAF is not a presentation format and is not an alternative to HLS or DASH. In fact, CMAF is deployed using either HLS or DASH. The efficiency of CMAF is driven by the fact that CMAF-wrapped media segments can be simultaneously referenced by HLS playlists and DASH manifests. This enables content owners to package and store one set of files, halving their costs for doing so. It also makes CDNs more efficient since the cache-hit rate doubles if the number of objects representing an asset is reduced by half. The standard also offers an additional intriguing benefit inherited from the DASH ATSC 3 broadcast profile: latency reduction, which is the topic of this paper.

With non-chunked encoding, since the Movie Fragment Box (“moof”) must reference all the video samples held in the Media Data Box (“mdat”), an encoder producing non-fragmented mp4 (CMAF) files must wait to encode the last byte of content before it uploads the first byte to the CDN for distribution. This introduces a delay (loss in latency) of one segment duration. In addition, CDNs receiving the incoming segments typically wait to receive the last byte before forwarding on the first byte, and players wait to receive the last byte from the CDN before beginning to decode the first byte. This pattern of repeated accumulation results in an overall latency loss that is an integer multiple of the segment duration. Delays of 5x segment duration are quite common in status quo deployments — ~10 s with 2 s segments and 20 s with 4 s segments— which still lag behind broadcast levels of latency.

What if we could break this pattern of accumulation? What if the encoded data was transferred down the distribution chain from encoder to client almost instantaneously as soon as the first frame of a segment was generated? The result would be a dramatic reduction in overall latency. A number of components must behave in concert to achieve this goal. Let’s examine these behaviors in turn.

Figure 1: CMAF object nomenclature
Chunked Encoding

The first required behavior is chunked encoding. Per the MPEG CMAF standard, a CMAF track is composed of a number of objects, as illustrated in Figure 1. A “chunk” is the smallest referenceable unit, containing at least a moof and a mdat atom. One or more chunks are combined to form a fragment, and one or more fragments are combined to form a segment. A separate header is required to initialize the playback of the chunks in the decoder. To clarify, just using CMAF segments themselves will do nothing to reduce latency. The latency will be identical to using standard DASH-ISO segments. To obtain low latency, the CMAF containers must be paired with encoder, CDN, and client behaviors so that the overall system enables low latency.

A standard CMAF media segment is encoded with a single moof and mdat atom, as shown in Figure 2. The mdat holds a single IDR (Instantaneous Decoder Refresh) frame, which is required to begin every segment.

A “chunked-encoded” segment, however, will hold a series of “chunks” (i.e., a sequence of multiple moof/mdat tuples), as shown in Figure 2. Only the first tuple holds an IDR frame. The advantage of breaking up the segment into these shorter pieces is that the encoder can output each chunk for delivery immediately after encoding it. There is no fixed rule for how many frames are included in each chunk. Current encoder practice ranges from 1–15 frames. Taking the example of a 4 s segment at 30 fps with 1 frame per chunk, the media content within the chunk is released 3.967 s earlier than if the encoder had waited to produce the last chunk of the segment before uploading the first. This early release leads to a direct reduction in overall latency by the same amount.

The extra moof/mdat tuples do increase segment size compared to conventionally encoded segments. An audio segment of 4 s duration at 64 kbps would normally have a size close to 32 kB. With chunked encoding of 1 frame/chunk at 30 fps, the segment size grows to 68 kB. The extra 36 kB is a relatively fixed overhead for that chunk frequency. While it leads to a 100% increase in audio segment size, it is only a ~2% increase for typical video segments due to their increased sample size. This overhead decreases linearly as the number of frames per chunk increases.
It should be clarified that CMAF did not “invent” chunked encoding. It has been available since 2003, when AVC was first standardized. The MPEG DASH ISO-based broadcast profile developed by the DASH Industry Forum for ATSC 3 standardized its use prior to CMAF adopting it. Chunked encoding has been used in many instances in academia and the video industry for over a decade. However, there is now a coordinated and widespread effort within the industry to use this approach for lowering latency.

Returning to our workflow, at this point the encoder does not know the final size of the segment it is going to produce. For HAS ingest, the encoder typically POSTs the segment to an ingest server. We now introduce our second required system behavior, which is chunked transfer encoding.

**Ingest and CDN Distribution**

A HAS media distribution system can be conceptualized as having two halves, as illustrated in Figure 3. In the contribution half, the encoder pushes (via HTTP POST) the segments to a live origin. A live origin has an ingest layer to accept the content and a mid-tier layer to present the content for distribution. On the distribution half, a player pulls the content (via HTTP GET) from an edge server, which in turn sources them from the origin. Both these halves need to work together to transfer the chunks as quickly as possible. To enable this, we deploy a relatively old technique known as chunked transfer encoding.

Chunked transfer encoding is a data transfer mechanism, available since version 1.1 of the Hypertext Transfer Protocol (HTTP), in which the server does not know the final size of the object it is sending. As a consequence, it does not insert a content-length header in the response and instead sends data as it becomes available in short bursts, called “chunks.” The transmission ends when a zero-length chunk is sent.

The encoder will use HTTP 1.1 chunked transfer encoding to send the encoded CMAF chunk to the origin for redistribution. To continue our prior example of producing 4 s 30 fps segments, it would make one HTTP POST every 4 s (one for each segment) and then during the next 4 s the 120 chunks that comprise the segment, each 33 ms long, would be sent down that open connection to the distribution cloud. Note that the encoder is not making a POST for each individual chunk.

The terms “chunk” and “encoding” are each being used twice here, once to represent the encoding and again the transfer. Since the nomenclature “chunked-encoded, chunked-transferred CMAF” is a bit of a mouthful, I am going to represent this combination as ULL-CMAF (ultra-low-latency CMAF), to separate this system from singular use of the CMAF file container itself.
Media Player

The remainder of the chunk’s journey is pull-based and driven by the media player. The media player reads the manifest or playlist, which describes the content, calculates the live edge at which it wishes to start playback (more on this later), and then makes a request for a segment.

To illustrate the sensitivity of overall latency to a player’s starting algorithm, Figure 4 shows a live encoder producing 2s segments. We observe the system at a point in time midway through the production of segment #5. A non-chunked solution could minimize its latency by starting with the last fully available segment (#4), resulting in 3s overall latency. If the content is chunked-encoded with 500ms chunks (for illustration, as chunks are actually much shorter than this), then the player could start with the latest chunk holding a keyframe (#5a), which would reduce the latency to 1s.

Two methods now exist to drop the latency even further. In the first, the player would download chunks 5a and 5 but then decode forward through 5a to 5b before starting playback, thereby lowering its latency to less than 500 ms. In the second, the player can defer playback by 1s and then make a well-timed request for chunk 6a immediately after it is produced, also reducing the latency to less than 500 ms. For both of these options, this segment request will occur within one segment duration of the true live edge, meaning that the encoder is still producing the last chunks of the segment while the client is requesting the first chunks. A legacy or conservative client, which is playing further back behind the live edge, will request the segment and receive the complete segment in the response, as is normal with HAS streaming.

Note that the player requests a segment and not a chunk, since the chunks are not addressable units (outside of ATSC 3, but that’s another case study). When asked for a segment, the CDN edge will return all the chunks it has for that segment in sequential order using chunked transfer encoding. As more chunks arrive from the origin, they are
fed to the client until eventually the complete segment has been delivered. Importantly, the CDN edge also caches the chunks flowing through it to build up a cached representation of the complete segment. This ability for a) CDNs to cache the complete segments and b) the stream to be backward compatible with the majority of clients that have not been optimized for low latency provides one of the strongest advantages for ULL-CMAF when compared to alternate schemas.

An interesting side effect of this chained chunked transfer is that the segments are delivered with consistent timing that is independent of the throughput between client and edge server. With traditional HAS, a 4 s segment might be delivered in just 1 s to a well-connected client, since the server has all the bytes that comprise the segment and the delivery time becomes the ratio of the throughput to the encoded bitrate. With chunked encoding and transfer, the edge server can’t send the bytes any faster to the client than it is receiving them from the server. The server produces 1 second of live content every second, hence every segment is sent in a time that approximately equals its segment duration. Figure 5 shows some real-world data in this regard. A 4 s segment is sent in ~3.8 s, and this time does not vary as long as last-mile throughput is greater than the encoded segment bitrate. This consistency is both a blessing and a curse for ULL-CMAF. The blessing is that the rate of bits crossing the wire at any point in the distribution workflow is very consistent. It has none of the sawtooth variability exhibited by traditional HAS. Traffic looks much as if a MPEG-Transport Stream were being used to multicast the content through the network. The curse is that HAS players use the segment download time to estimate last-mile throughput, which is an integral metric for adaptive bitrate switching and maintaining a consistent experience for the viewer. When using a chunk-based approach with CMAF, a standard throughput estimation algorithm will produce the answer that the connectivity is exactly equal to the encoded bitrate, which will prevent the player from switching up. Various workarounds exist for this problem, including measuring the connectivity as the chunks are burst and then applying a conservative average, as well as machine learning to infer connectivity given a pattern of chunk burst times. Players need to be taught this new behavior if they are to play back multi-bitrate ULL-CMAF segments successfully.

A second required behavior of a ULL-CMAF player is catch-up. In low-latency applications, the player buffer will be small — typically one segment duration or less. An end-to-end-latency of 3 s might be taken up by a 500 ms encode time, a sub-500 ms CDN propagation time and then a 2 s player buffer. The goal is to extract all latency out of the CDN, leaving it in the encoder — where additional latency increases quality, and in the player, where latency protects against distribution perturbations.
Since the likelihood of an errant segment is real, most low-latency players include a catch-up feature, which pulls the player back to the optimum live edge using two methods (often in combination):

1. **Rate control** – By playing content at a rate greater than the encoded rate, the play head will move forward in time. Conversely, by slowing playback, it will be delayed. Add in a control loop, and a player can hold itself pretty tightly to a target latency by controlling playback rate. Depending on the content, variations in rate of 5% or less are not noticeable to end users, especially in browser implementations in which you receive automatic pitch correction for the audio. Figure 6 shows a plot of the live latency as a well-performing player pulls itself forward to its target latency of 2.5 s.

2. **Jumping forward** – At some point the player may fall behind live to a point at which rate control adjustments provide an unpleasant viewing experience. It can then implement a jump forward in play head position. This will present a discontinuous experience to the viewer, but it is preferable to falling far behind. Back-off logic must be written to prevent players with insufficient bandwidth from constantly cycling back to live.

A third required feature of a ULL-CMAF client is accurate timing. Any delay between a segment being available at the edge and a client requesting it translates directly into additional latency. How does the client know when to request the segment? The answer differs depending on whether HLS or DASH is being used to describe the stream. In MPEG DASH, the manifest signals the early availability of the segment via the MPD@availabilityTimeOffset parameter. Since there is no explicit timing model in HLS, the client must poll the server and wait to receive an updated child-playlist before it can request the segment. This means that segments must be advertised ahead of time in the playlist and the edge servers must hold early requests for content open until the data arrives from the origin.

**Workflow Summary**

To reiterate, stable latency reduction with ULL-CMAF is only achieved if all of the following are in place:

a. The content in the CMAF segment is chunked-encoded.

b. The encoder adjusts its manifest.playlist production to accommodate and signal the usage of chunked encoding and early availability of the data.

c. The encoder pushes content to the origin using HTTP 1.1 chunked encoding transfer.

d. The CDN propagates this content all the way to the client using HTTP chunked encoding transfer at each step in the distribution chain.

e. The client:

   1. times the request for the segment accurately and requests the segment within one segment duration of the live edge;

   2. decodes the bitstream as it is received and does not wait for the end-of-segment. HTML5 players operating in a browser must use the Fetch rather than the XHR API, as Fetch allows for reading the response body while the data is still being downloaded;

   3. has a scheme to estimate throughput as standard segment-timing techniques will fail;

   4. has buffer and adaptation logic to cope with very low buffers;

   5. has a catch-up feature should it fall behind live due to fluctuations in throughput.
Operating Profiles

It is possible in the lab to use ULL-CMAF to produce glass-to-glass latencies in the 600 ms range. These are excellent for impressing friends and CEOs, however they become increasingly fragile with scale and with geographic dispersion (i.e., higher round-trip times between encoder, origin, edge server, and clients). If distribution is happening over the open Internet (especially over a last-mile mobile network where rapid throughput fluctuations are the norm), current proofs-of-concept show more sustainable Quality of Experience (QoE) with a glass-to-glass latency in the 3 s range, of which 1.5–2 s resides in the player buffer. Current encoder implementations tend to favor one video frame per chunk, but there is no objective data yet to indicate whether this is optimum from a robustness or quality perspective. I expect the academic community to pursue this question with vigor.

LHLS

What about low-latency HLS, you ask. Is this not similar to what is being described here? Indeed, it is. LHLS is a low-latency implementation of HLS, as implemented by Twitter in their Periscope product. In this schema, TS segments are produced by the encoder. These are naturally chunked-encoded, due to the native file structure of TS containers. Chunked HTTP transfer is then used to send these segments through the CDN and forward to the client. Being HLS, the encoder must advertise segments before they are produced, and the edge server must co-operate and hold early requests open until the data is available. The disadvantage of TS versus CMAF containers is that they are not supported by MSE implementations and must therefore be dynamically repackaged into ISO containers by client software such as HLS.js. This repackaging operation can be avoided if ULL-CMAF is used. Over time, we can expect LHLS to adopt CMAF segments, and therefore for the ULL-CMAF and LHLS workflows to converge.

Benefits

Chunked-based encoding and transfer is a viable solution for low-latency delivery to end viewers across a range of use cases, including OTT. Other options do exist for low- and ultra-low-latency systems, specifically UDP-based proprietary solutions as well as solutions based on WebRTC. While these solutions all have their respective benefits, the advantages of chunked-encoded, chunked-transferred CMAF are:

1. Legacy player support – Players that are ignorant of ULL-CMAF will still play the streams, albeit at a higher latency and assuming that their decoders can handle chunked encoding.

2. Decoupling of latency from segment duration – with chunked encoding and chunked transferred CMAF, the same latency can be obtained from 6 s segments as from 1 s segments. And long GOPs can be used, in order to maximize quality per transferred bit.

3. CDN cacheability – Players playing more than one segment duration behind live receive the fully cached segments. This means that for a given live ULL-CMAF stream, players can choose to position themselves at different points on the latency/robustness curve. For example, a provider could charge a premium for clients which sit closer to live, while delivering the same stream to non-premium clients.

4. CMAF supports common encryption – To minimize a latency penalty for this, players should request keys ahead of joining the stream, since key retrieval can be a slow synchronous process.
5. **Retention of UHD Features** – ULL-CMAF, being CMAF based, supports the latest codecs allowing encoders to retain UHD features such as 4K, HDR, and high frame rate, with the caveat that using these features may increase encode time. All these features increase the visual quality of the picture being transferred.

6. **Codec diversity** – You can offer ULL-CMAF with any codec that presents a CMAF media profile, such as AVC, HEVC, and AV1 (assuming you can find a live encoder for the latter).

7. **Monetization** – Since ULL-CMAF does not change the manifest/playlist structure of adaptive streaming, it retains compliance with existing server-side ad insertion (SSAI) networks.

8. **Standards-based** – CMAF is defined by MPEG and HTTP 1.1 by IETF RFC 7230, so it does not rely upon custom protocols or proprietary systems of encoders and players, which can result in very costly architectures and strong vendor lock-in. The ability to replace an encoder, a CDN or a player by a more performing one or a cheaper one at equivalent quality is a key success factor in the OTT world.

9. **CDN scalability** – Since ULL-CMAF leverages 21-year-old HTTP technology, it is widely supported in the HAS infrastructure of modern CDNs and can leverage their existing cache hierarchies to achieve scale. If you need to support millions of viewers on your low-latency stream, ULL-CMAF is a strong technical choice.

**Challenges**

There are some acknowledged challenges to deploying chunked-based CMAF. The DVB (Digital Video Broadcast) organization, who define broadcast standards for much of the world outside of the United States, are looking at standardizing ULL-CMAF to provide low-latency streams for DVB-DASH, in which case we can expect future TV sets to be compliant. However today these deployed TVs represent a non-addressable legacy market for ULL-CMAF, as they may not support chunked encoding.

A second problem occurs around advertising. While ULL-CMAF is compliant with server-side ad insertion (SSAI) HAS workflows, these have been designed for legacy OTT solutions in which longer player buffers allowed for a several-second time period in which to make the placement decision. With low latency, that luxury of time goes away, and many modern SSAI solutions will find themselves unable to react sufficiently quickly. There are ways to solve these problems, such as by pre-resolving and pre-fetching ads, but any difficulty in monetizing content can provide a hurdle to adoption. Note that this hurdle is common to any solution exploring low latency and not particular to ULL-CMAF alone.

A third issue to consider is lack of standardization and support for embedded EMSG boxes. DASH IF interop guidelines indicate that EMSG box can be placed only in the first chunk of a segment. Compliance is simple with non-chunked-encoded content, as the encoder can go back and insert the EMSG box if the call for the event occurs anytime over the life of the segment creation. With chunked-encoded content however, the first chunk has already been dispatched for distribution, so subsequent events would need to be held and placed at the start of the next segment, which delays their detection by the player. This problem could be solved by allowing EMSG to be placed in any chunk, but the player decoder pipeline would also have to be extended to look for EMSG boxes in any chunk, which is a more expensive parsing operation.
Deployment

Overall, the benefits of ULL-CMAF outweigh the risks, and many vendors are already moving forward with solution. As of Q3 2018, here are some workflow components in production:

1. **CDN** – Akamai Media Services Live (MSL) origin can accept chunked-encoded POST to the ingest as well as chaining that transfer through the mid-tier and forward to the Akamai Media Delivery (AMD) edge servers. CDN transit times from ingress to egress of less than 500 ms are possible in large-scale production by default today.

2. **Players** – The open source dash.js project supports ULL-CMAF as do commercial players from Akamai, castLabs, NexPlayer, and THEOplayer. The list of compatible players is growing at every trade show. Players can be found for iOS, Android, HTML5, and other application environments.

3. **Encoders** – Anevia, Cisco, GPAC Signals, Harmonic, and Media Excel. Akamai has also contributed code to FFmpeg to allow this open source project, available under the GNU Lesser General Public License, to publish ULL-CMAF using both DASH and LHS manifests.

Demo

A public working demo is available for viewing in the Chrome browser. This demo shows a live stream produced by open source FFmpeg, publishing into Akamai Media Services liveOrigin, delivered over the Akamai Media Delivery and played back by the dash.js open source player. Stream is AVC 1080 p at 4.5 Mbps with a segment duration of 6 s, chunk duration of 1 frame at 29.97 fps and availabilityTimeOffset is signaled at 5.967 s. Target latency is set to 2.8 s and the stream is being encoded in Boston.

What’s Next

All components of the ULL-CMAF system are seeing broad improvement. The advantage of a standards-based approach to low latency is that the resources of many companies are combined for mutual benefit. Encoders are becoming more proficient at encoding chunked content, studies are being done to establish chunk-size/robustness/quality curves, and UDP protocols such as QUIC are being investigated for last-mile delivery. Standards bodies are looking at issues around EMSG, and we will see SSAI vendors enable solutions in low-latency environments. As standardization and commercial models push forward, OTT continues to aggressively challenge broadcast norms for quality and latency. The quality-at-scale target inches ever closer.