Title	Real Time Segmented Streaming
Author	Jonathan Valliere
Copyright	2019 Jonathan Valliere / Look At Me, Inc.
Date	July 20, 2019
Version	0.3
Status	Draft Proposal

Notice

This is a preliminary specification, and it is being provided in an effort to establish open development of a specification for producing and consuming real-time Media through existing Content Delivery Networks.

This specification does not attempt to define a playlist format because it is desirable to work with other engineering groups to prevent duplication of efforts and promote cross-compatibility.

A Project Management Committee, consisting of industry members, is intended to be established to oversee the future work of developing and promoting the final RTSS specification.

While the attached preliminary specification and all images contained therein are copyrighted, the intention is that the author will be part of the intended Project Management Committee which will, upon completion of its work, prepare and release to the public the final RTSS specification. It is the author's expectation that rights to the final RTSS specification will be determined by the Committee with the permission and consent of the contributing parties.

It is specifically intended that the Committee and all persons working with, on behalf of, or for the benefit of the Committee, shall have open access to the materials in the attached preliminary specification, and that the use of any copyrighted material, herein, for any purpose, other than for the purposes of the Committee, is reserved to the author.

Jonathan Valliere, author

Real Time Segmented Streaming

Segmentation format for large scale real-time delivery of multimedia presentations through an object caching infrastructure.

Table of Contents

Figures

1 Introduction

Since moving from real-time frame-based push live streaming to a segment-based pull live streaming, the industry has been unable to reproduce the same latency as push while maintaining compatibility with Content Delivery Network (CDN) caching systems. This document outlines a common segment format for real-time segmented streaming over HTTP.

1.1 Scope of Document

This document outlines the segment format and compatibility points for developing interoperable players and servers capable of real-time segmented streaming.

RTSS is intended to be independent of the manifest/playlist format. Existing manifest/playlist-based streaming protocols will require signaling to indicate compatibility with this specification. Such signaling will be defined in future amendments to this document.

1.2 Status of Document

This specification is a work in progress, it is based on experience developing a prototype system which uses multiple track segments. More work and understanding is required to express track synchronization across multiple segment streams with discontinuities.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

These normative references are intended to include corrigenda and amendments available at the time of use.

[DASH]	ISO/IEC 23009-1, Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats
[HLS]	RFC8216, HTTP Live Streaming *https://tools.ietf.org/html/draft-pantos-http-live-streaming-23*
[CMAF]	ISO/IEC CD 23000-19, Common Media Application Format for Segmented Media
[ISOBMFF]	ISO/IEC 14496-12:2015, Information technology – Coding of audio-visual objects – Part 12: ISO Base Media File Format
[MP4RA]	Registration Authority for the ISOBMFF family of standards
[M3U8]	Moving Picture Experts Group Audio Layer 3 Uniform Resource Locator *https://en.wikipedia.org/wiki/M3U#M3U8*

3 Terms, Definitions, symbols and abbreviated terms

For the purpose of this document, the following terms and definitions apply.

ABR	Abbreviation for Adaptive Bitrate. Adaptive Bitrate is a method of switching between multiple qualities (High Definition, Low Definition) of a single stream depending on how well the User's internet connection is functioning.
Broadcast	Transmission of Media from point A to point B
Client-Side	Occurring on a Client
Coordinated-Clock	A method of using a local clock and a remote clock to determine an offset used in conjunction with the local clock to determine the current time at the remote clock. This method is used in DASH-LIVE to predict segment availability without having to poll for playlist changes.
Delay Floor	The minimum technical Playback Delay to achieve uninterrupted playback.
Encoding Delay	The delay between the time a Media Frame is captured by the Media Capture and transferred to the Segment Encoder.
Low-Latency Stream	A stream which has a Playback Delay of under 6 seconds
Media	Audio / Visual / Subtitle Data
Media Availability Delay	The time between the moment a Media Frame is received by the Segment Encoder and the moment it is made available for download by a Player.
Media Capture	A hardware / software component which captures raw Media from devices such as Cameras and Microphones.
Media Encoder	A software component which captures Media and compresses it into a suitable format for packaging with the Segment Encoder
Media Frame	Unit of Audio / Video / Subtitle Content
Movie Fragment	Defined by [ISOBMFF] as a combination of a `moof` and `mdat` box.
Playback Buffer	The amount of pending Media Frames stored in the Player represented as a duration of time.
Playback Delay	The delay between the time a Media Frame is received by the Segment Encoder and the same Media Frame is rendered in the Player
Player	A software component for rendering Media Frames
Playlist	A structured document containing URLs for one or more Media Segments
Real-Time	Occurring without or with an insignificant delay
Real-Time Latency Stream	A stream which has a Playback Delay of under 1 second

Round Trip Request	The time it takes for a single request to be sent to a remote device and the response to be received from said remote device.
Segment	A structured document for containing Media Frames generated by the Segment Encoder
Segment Encoder	Software component which packages Media Frames into Segments.
Segmented Stream	Sequence of Segments used to produce a continuous Stream
Server-Side	Occurring on a Server
Stream	Sequence of sequential Media Frames
Streaming	Process of receiving and playing Media Frames
Ultra-Low Latency Stream	A stream which has a Playback Delay of under 3 seconds
User	Usually a person who is operating the Player
User-Agent	An intermediary software component which issues HTTP Requests and manages TCP connections on behalf of the Player.

4 Challenges of Low Latency Segmented Streaming

Segmented Streaming was not originally designed for Low Delay functionality; the use of asynchronous updates to the stream playlist meant that many changes to the attributes of the stream required a hard delay floor of several seconds to reduce playback stalls. Technologies such as DASH-LIVE introduce new concepts to remove the playlist update from the stream playback workflow but are completely negated during any abnormalities or features such as advertising-insertion. The following section describes some causes of stream delay.

4.1 Effect of Segment Duration on Playback Delay

In traditional HLS/DASH the Segment Encoder must delay the stream by the target segment duration before making segments available for playback. In order for Players to maintain enough internal buffer to prevent stalls, Players must use an internal buffer having a multiple of the Segment Encoder delay. In the following figure "Segment Encoder Delay" is the same as "Media Availability Delay".

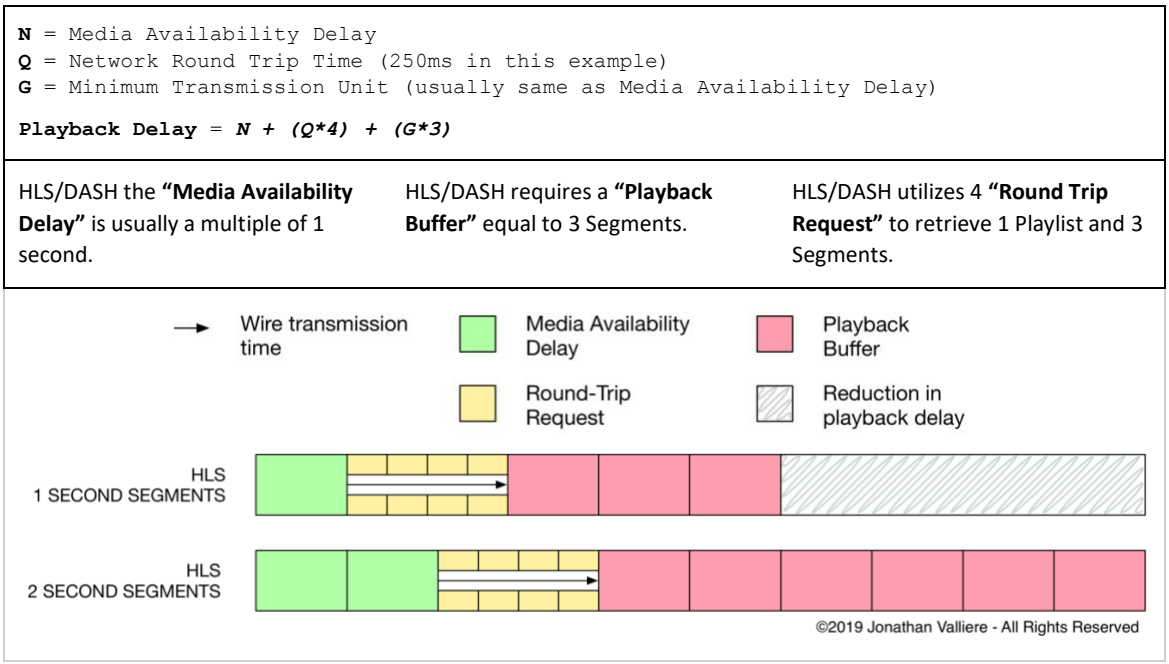

Figure A - Playback Delay by Segment Duration

4.2 Effect of Segment Duration on Encoding Efficiency

Bitmovin, Inc. released an analysis of PSNR (Peak Signal to Noise Ratio) utilizing different GOP (Group of Pictures) distances, ultimately deciding that the benefit of larger GOP distances becomes significantly reduced as the GOP reaches 4 seconds. See https://bitmovin.com/mpeg-dash-hls-segment-length/ for more information.

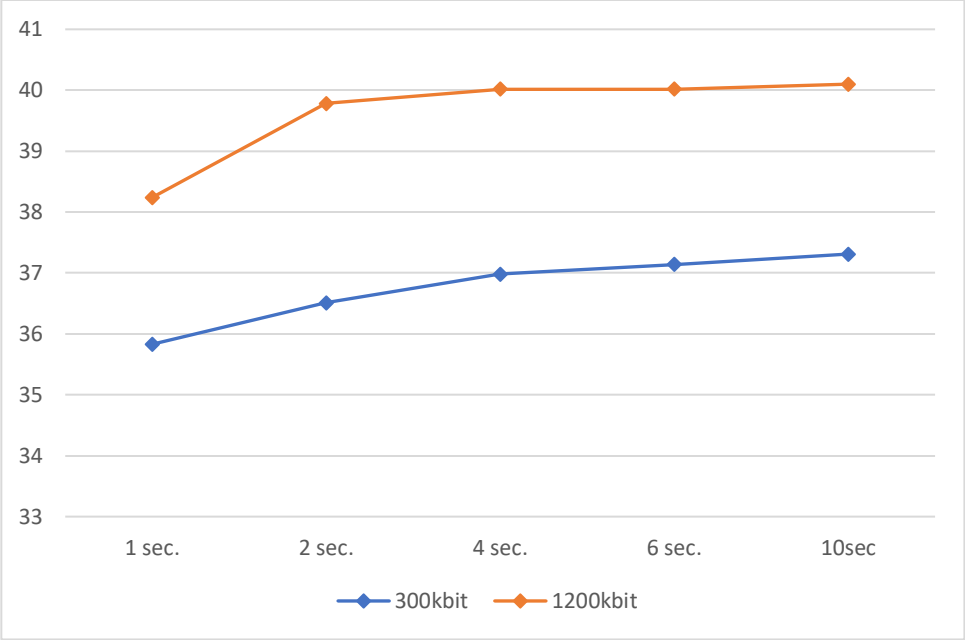

Figure B - PSNR Chart

4.3 Effect of Packet Loss on Segment Availability

4.3.1 Lossless Broadcasts

Lossless broadcast protocols such as SRT, RTMP and HTTP PUSH (Linear Blocking) are lossless by design. However, due to bandwidth constraints, many broadcasters have the ability to adapt the bitrate and/or drop frames to maintain throughput. Bandwidth constrains will negatively affect the timeliness of media delivery.

Figure *C* represents the negative impact packet loss and the subsequent recovery delay of the segment creation cadence on a remote device. This negatively impacts the ability to use coordinated-clock segment prediction such as defined for DASH-LIVE. When attempting to achieve ultra-low latency, the player and delivery network may produce 404 error messages due to this recovery delay.

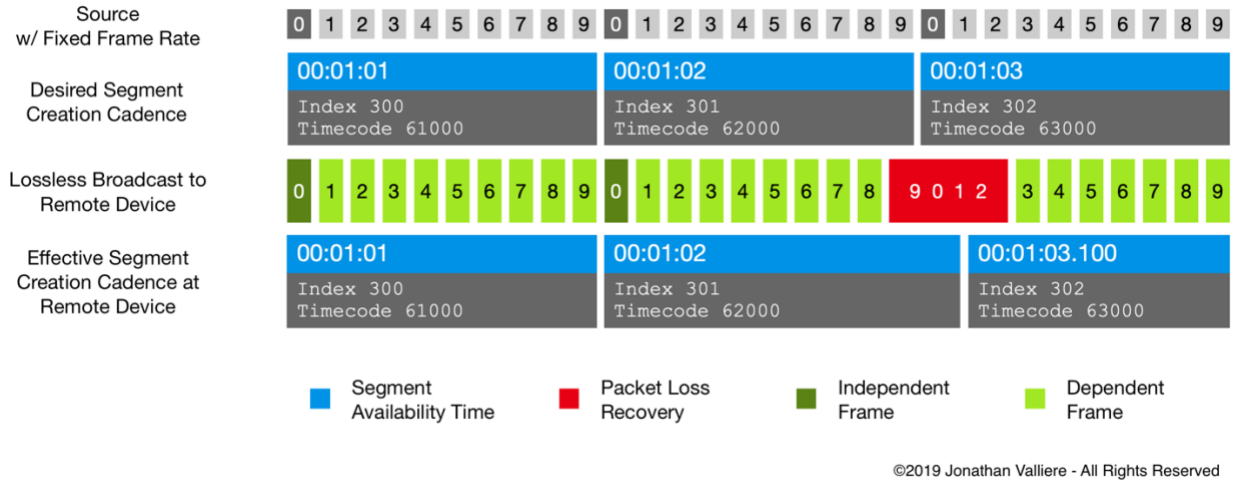

Figure C - Lossless Broadcast Segment Cadence

4.3.2 Lossy Broadcasts

Lossy broadcast protocols such as RTP or SRTP, having limited data recovery measures, pose issues when converting the stream into segments. In situations where data may not be recovered, segment durations and availability may shift from expected cadence.

Figure *D* represents the impact that non-recoverable packet loss on the segment creation cadence on the remote device. In this particular situation, using the coordinated-clock to predict segment URLs using the index values will break the stream playback due to a Segment Index 300 under "Most Common Segment Creation Cadence" is much longer than expected. For protocols like DASH-LIVE, the timecodes are more reliable for prediction. The difference between "Best Possible ... Cadence" and "Most Common … Cadence" is that the "Best Possible" violates the rule that all segments must begin with a keyframe. **Figure *D*** represents a situation where 404 errors would be produced when requesting Index 301 or Timecode 62000 at the expect time of 00:01:02-00:01:03. Most likely this would produce playback stalls across the board unless the player buffer is large enough to compensate.

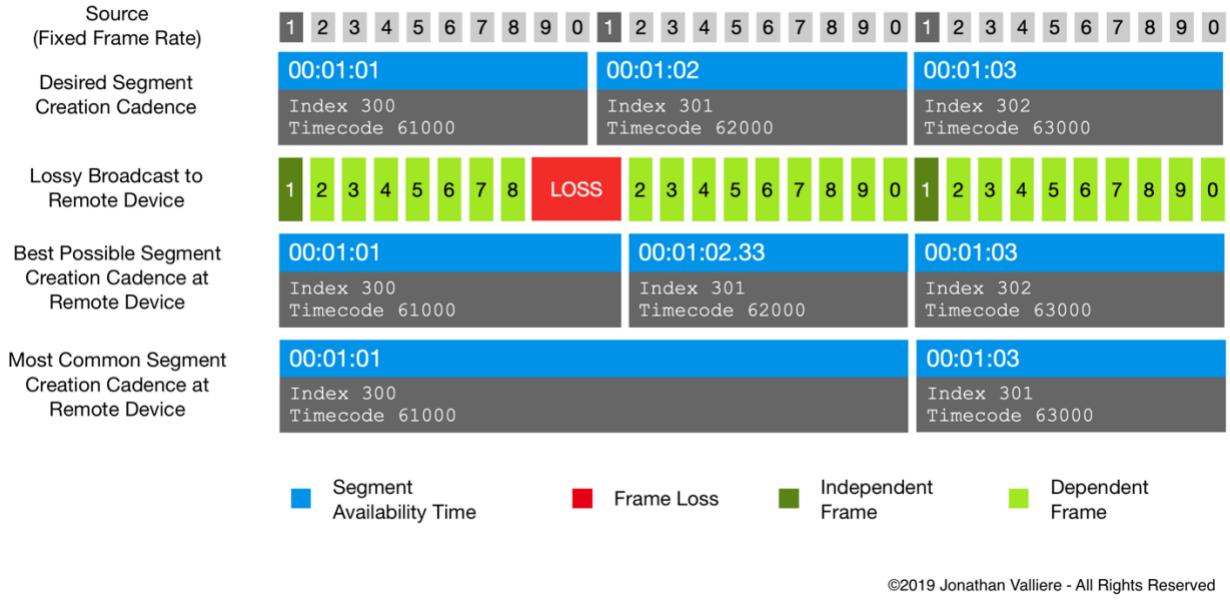

Figure D - Lossy Broadcast Segment Cadence

4.3.3 Conclusion

Gaps in media content, due to dropped or lost data, will prevent accurate calculation of segment durations when buffered media content must be made available to prior to receiving media for the subsequent segment prevent delays.

While it is possible to design some processing systems that are able to negate some of these issues, there becomes a much larger problem with compatibility with different broadcaster sources and a potential latency tradeoff.

As described in this specification, the use of cascading ahead-of-time segment requests provides a means to maintain ultra-low latency playback for segments with variable durations. Thereby fully mitigating the latency inducing effects of packet loss on the segment creation cadence.

4.4 Effect of HTTP Caching on Periodic Playlist Refresh

4.4.1 Introduction

In this section, the effects of rounding of DATE fields of documents delivered through a HTTP Caching infrastructure is discussed.

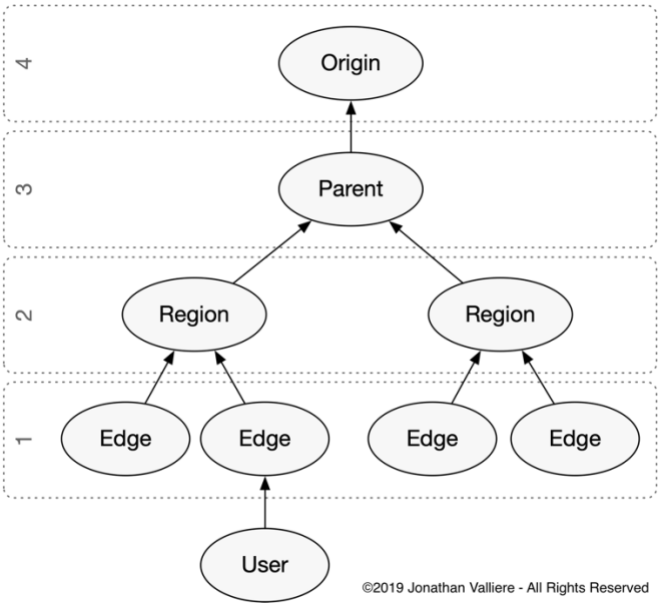

Figure E - HTTP Caching Tiers Example

Both HTTP AGE and HTTP EXPIRES header fields are limited to 1 second precision. To produce these values, some form of rounding is required. Neither RFC 5176 nor RFC 7231 define the desired rounding algorithm causing additional situations where documents may be held in caches much longer than desired. When used in low-latency live-streaming, documents having desirable expirations equal to or less than the 1 second precision causes significant problems with delivering fresh responses to the Player further increasing the Delay Floor.

4.4.2 Using HTTP AGE

The HTTP AGE method of expiring documents produces a relativistic expiration having the ability to be both more and less accurate than using EXPIRES depending on the scenario. The 1 second precision of HTTP AGE and the compounding rounding errors creates situations where documents are held much longer than desired as requests are propagated through the Delivery Network.

As shown in **Figure F**, the right-most request in Tier 2 produces a “Calculated Expires 00:01:32.300” of nearly 800ms longer than the “Document Expires 00:01:31.633” in Tier 4. This compounding problem becomes more significant when more caching tiers and the User Agent become involved.

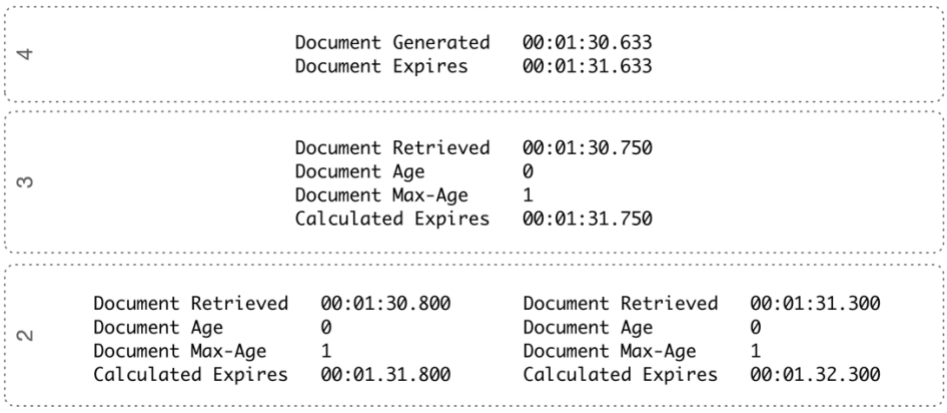

Figure F - HTTP AGE Compounding Errors

4.4.3 Using HTTP EXPIRES

The HTTP EXPIRES method of expiring documents produces a coordinated expiration time which is more predictable across multiple tiers of a Delivery Network than HTTP AGE, assuming that the device clocks are synchronized. Like HTTP AGE, HTTP EXPIRES rounding may cause documents to be held in caches longer than desired. However, the maximum "overtime" is a limit of the date rounding precision and not a compounding problem like in HTTP AGE. Unlike HTTP AGE, HTTP EXPIRES may produce a situation where documents are set to immediately expire (due to rounding errors).

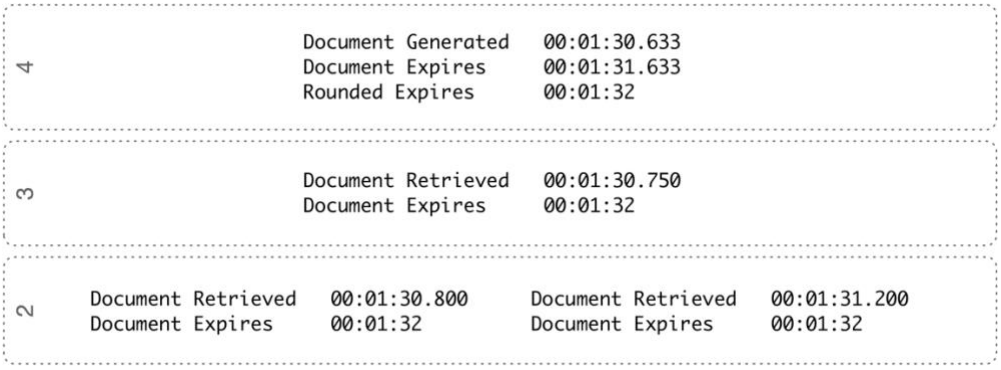

Figure G - HTTP EXPIRES (Rounding Up) held longer than desired

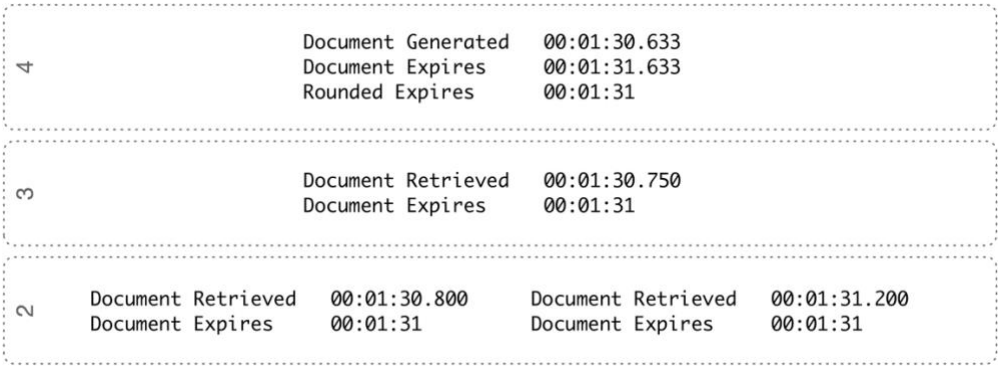

Figure H - HTTP EXPIRES (Rounding Down) held shorter than desired

4.4.4 Conclusion

Content delivery networks are generally not well suited to high precision expiration of cached resources due to the limited precision of the date formats in HTTP. For this reason, it becomes more likely to have a premature cache response of the same (old) playlist, causing ultra-low latency streams to stall, or significant decreased caching efficiency due to non-cacheable response which expire immediately. Therefore, using HTTP caching with short-lived documents is problematic for low latency streaming because the effective stream latency becomes hindered by the retrieval time and the caching expiration accuracy.

The mitigation of playlist documents described in this specification entirely removes the caching complications described above while supporting ultra-low and sub-second latency.

4.5 Effect of Inconsistent Stream State between multiple Point of Presence

Intelligent Content Delivery Networks may provide stream replication mechanisms across multiple regions in order to balance and optimize inter-network traffic. As with any replication system, there exists variable delays between replication of all the individual components.

These variable delays create situations where a Viewer is aligned with the state of a specific region (HTTP Component C) and having intelligent redistribution cause requests from a Viewer to go to a different region (HTTP Component E) having a slightly different state than the previous region. In situations such as this, it is possible to encounter false-positive error signals when requesting a resource identified in the state of the first region from an inconsistent second region. Therefore, it is desirable for a Viewer to be capable of transitioning from one region to another without increasing the overall stream delay or causing an interruption in playback.

The techniques described in this specification, allow transitioning between regions by providing a dynamic look-ahead window described in **Section 8.2**.

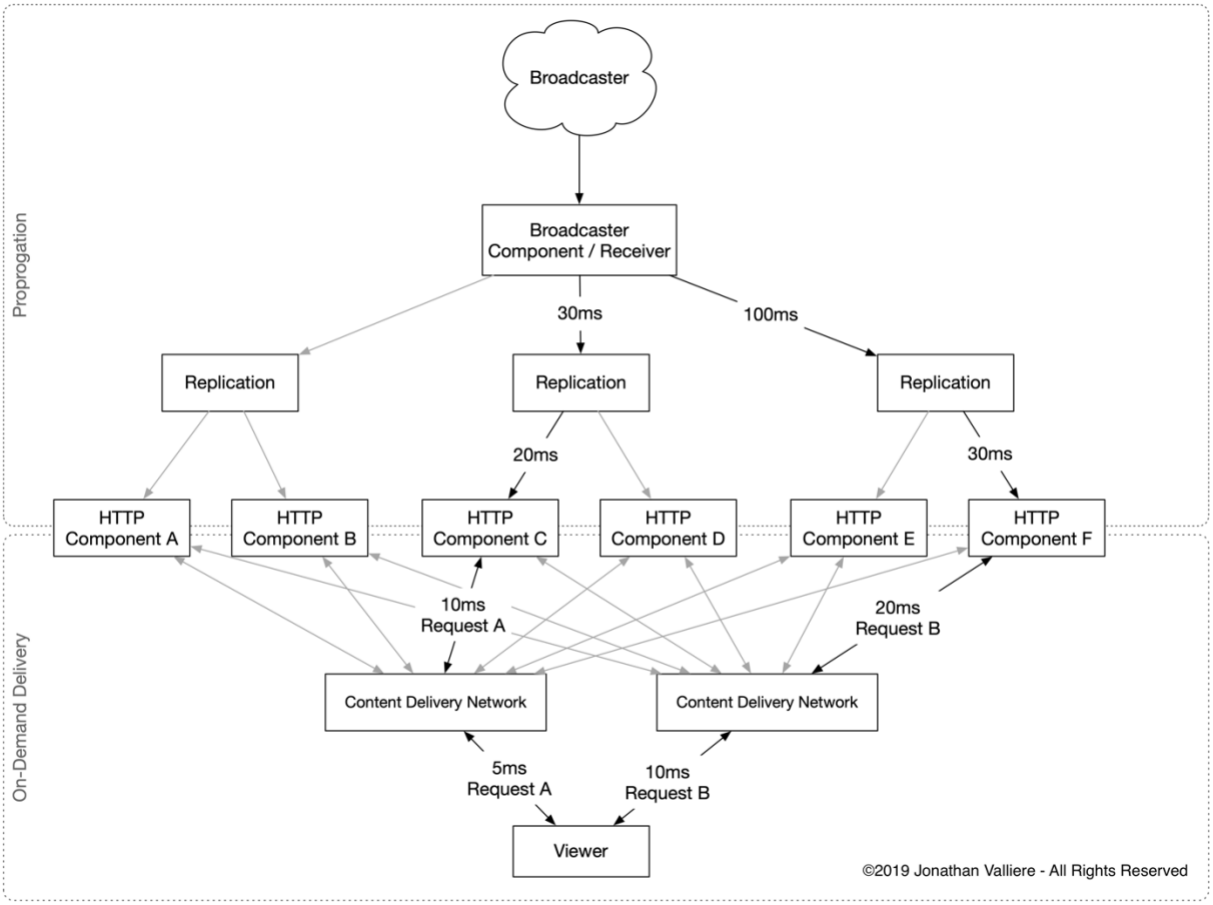

Figure I – Inconsistent Stream State across Multiple Point of Presence

5 RTSS Overview

5.1 Introduction

RTSS uses ahead-of-time segment requests to deliver consistent screen to screen latencies as low as 300 milliseconds in HTML5. In order to promote cross-compatibility, RTSS does not define a manifest / playlist format; instead relying on amendments to existing protocols such as HLS and MPEG-DASH to provide players with the necessary information to bootstrap RTSS formatted streams.

5.2 Architecture

This specification describes functionality assuming the following generalized architecture. **Figure J** describes the components of the architecture, these are not necessarily separate devices. For the purposes of simplification, the User Agent is absorbed by the Player box below.

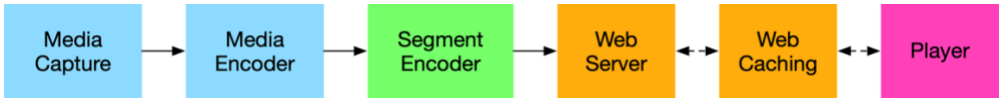

Figure J - General Architecture Diagram

5.3 Functional Goals

In the past, real-time or relative-real-time methods of live streaming required a consistent one-to-one connection between the Player and the Server. This style of streaming has obvious scalability issues but also limited error recovery as interruptions between the User and that specific Server break down. The design of RTSS seeks to provide similar latency as one-to-one without any of the possible negative effects of the design.

- Maintain similar overall Playback Delay as a continuous one-to-one streaming, such as RTP.
- Scalability achieved through Content Delivery Networks
- Real-Time transfer of Media Frames from Segment Encoder to Player through Middleware such as Web Caching.
- Real-Time Signals for seamless stream transitions such as SCTE-35, Client-Side Advertising Insertions, Discontinuities, etc.
- Transparent balancing of User Agent requests to multiple point-of-presence (POP) and Regions without interruption of playback.
- Continuous playback of a stream without requiring just-in-time periodic playlist updates overcoming the issues described in **Section 4.4**.
- Cascading Segment Requests enabling arbitrary segment durations overcoming the issues described in **Section 4.3** and increasing Media Encoder efficiency by utilizing variable keyframes.
- Fast Recovery from Broadcast or Middleware interruptions
- Backwards compatibility with traditional segmented streaming protocols.
- Maximize compatibility with existing media capture, media encoder, and broadcast components.

5.4 Streaming Segments

Streaming of Segments is a process for allowing Players to request a Segment before said Segment is completed and receive portions of the segment as they become available. This process significantly reduces overall playback delay compared to traditional segmented streaming by disassociating the playback delay from the target segment duration. This means that a 4 second segment and a 1 second segment end up with the same exact Playback Delay allowing for ultra-low latency streaming of high-quality video. See **Figure A** for an explanation of **Figure K**.

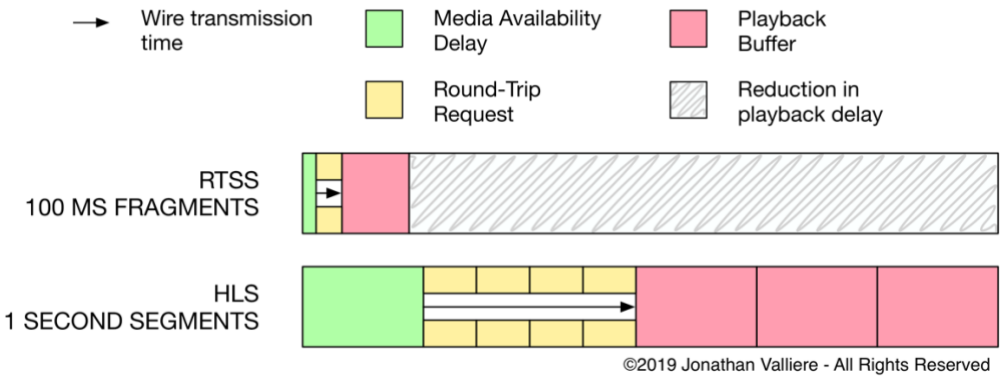

Figure K – Improvement of Playback Delay using Streaming Segments

5.5 Streaming Signals

RTSS utilizes the RTSB box, defined in **Section 6.3.2.1**, to notify Players of changes to the stream. The RTSB box is generated by the Segment Encoder and is contained within Segment documents. When combined with Streaming Segments, events included in RTSB boxes are received by Players in real-time.

5.6 Cascading Segment Requests

Cascading Segment Requests is a process in which the next segment request occurs immediately after the previous request completes. This allows for existing Media Capture and Media Encoder components to produce ultra-low latency streams without the problems described in **Section 4**. Additionally, Cascading Segment Requests enables the use of variable Segment Durations increasing potential Media Encoder efficiency and Video Quality.

For RTSS, there are usually two pending requests for every Media Presentation in order to allow for instantaneous transition between the request completing and the next request just starting. This allows the Player to utilize multiple segment requests in a way as to achieve the same delay as a real-time stream such as RTP or RTMP. An example of the request waterfall is shown in **Figure L.**

When combined with Streaming Segments the Player can utilize the download speed relative to the media duration to determine if media is being delivered faster than real-time. This allows the Player to maintain tracking of the relative stream delay.

Utilizing Cascading Segment Requests, RTSS is by far the most flexible Segmented Streaming Protocol only requiring a compatible Segment Encoder. That old IP Camera, in your basement, which drops frames and is otherwise produces unpredictable streams, could produce an ultra-low latency stream using RTSS from the built-in RTP broadcast.

5.7 Ahead-of-Time Segment Requests

Just-in-time request of segments requires coordinated-clock when deployed at scale and does not provide any means of establishing relative operation of the stream causing simple interruptions to introduce catastrophic failures when the Playback Buffer is less than 2 seconds.

Ahead-of-time provides additional redundancy by issuing requests at a time it is not critical to complete quickly. This allows for standard network operations like DNS resolution, TCP connections, and TLS negotiations to occur without negatively impacting on the transfer of real-time data. Have you ever seen a DNS resolution take 400ms? Ever had a CDN tier take 800ms to establish an upstream connection due to packet loss or network contention? There are a lot of operations that can go wrong at any point in time, more so as the systems between the User Agent and Origin Server become more complex. *Ahead-of-time* is the most resilient and optimal method of reducing latency while maintaining as much compatibility and tolerance for undesirable behavior to occur.

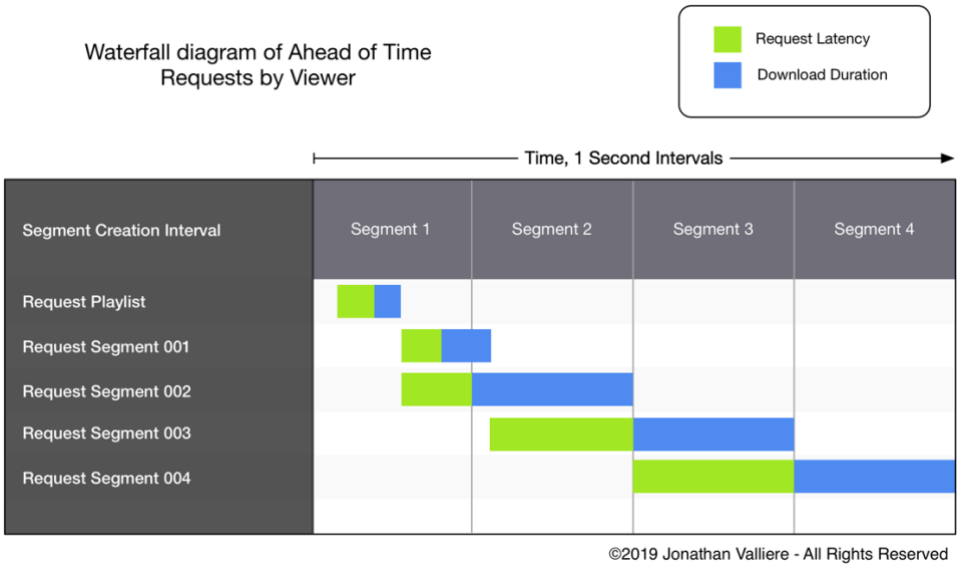

Figure L - Concurrent Request Waterfall Diagram

Additionally, ahead-of-time is simpler to operate and more stable over long playback sessions.

5.8 Fast Recovery from Broadcast interruptions

RTSS compatible servers provide a method of waiting for stream content to become available without producing error messages and since streams do not progress while interrupted, Players can simply poll-and-wait on the next segment in the stream without bogging down the Origin. Middleware components will limit how long the Player may wait; however, a simple retry brings the Player in synchronization once again.

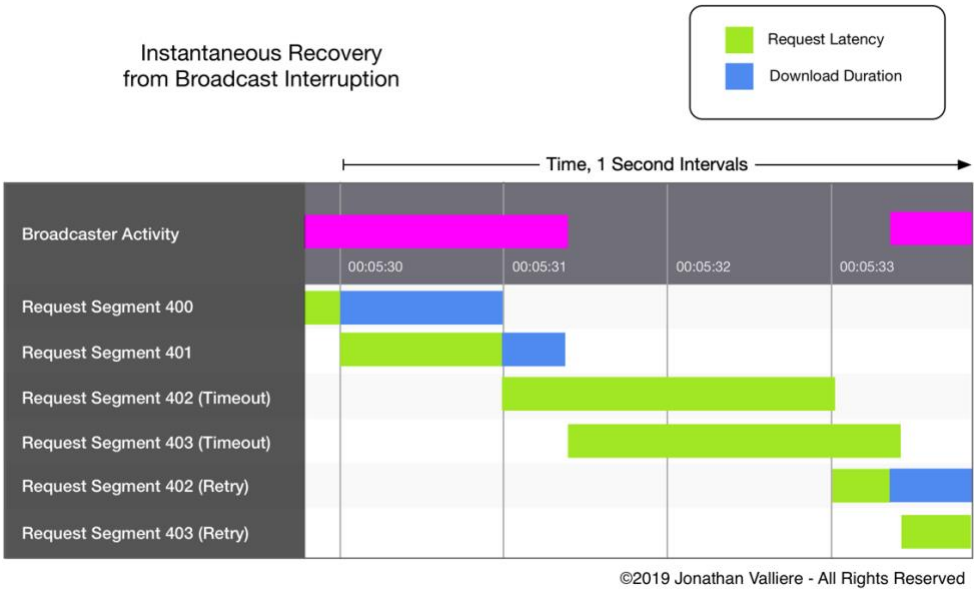

Figure M - Instant Recovery from Broadcast Interruption

When the source signal recovers, the Player can utilize a reference clock to determine if media content is being delivered faster than real-time; the Player can simply fast-forward through the stream to obtain synchronization with the broadcaster source. This prevents broadcaster backlog flush from causing all

Players to become delayed from target latency. In many situations, this recovery should be imperceivable to the User.

5.9 Interactions with Content Delivery Networks

Content Delivery Networks (CDN) can introduce a wide range of volatility of response times when obtaining fresh resources from the Origin server. Since CDNs are required to achieve scale, the streaming protocol must anticipate the use of CDNs and mitigate common issues.

RTSS negates the volatility of response times of the CDN by shifting common time delays to moments before data is expected to be available allowing streaming to occur in real-time.

RTSS negates the negative effects of HTTP Caching as described in **Section 4.4** by using cascading ahead-of-time segment requests instead of requiring the playlist to be updated every second.

5.10 Compliance with other Specifications

5.10.1 CMAF (Common Media Application Format)

RTSS is generally compatible with the restrictions set forth in [CMAF] unless a specific incompatibility is desirable. For example, [CMAF] restricts a single Media Codec for every Media Presentation. RTSS does not enforce this restriction, however specific Players may require this restriction.

5.11 Comparison with other Protocols

Previous attempts at creating low latency streaming using Periscope LHLS or DASH Live Profile come with serious disadvantages when compared to traditional segmented streaming.

5.11.1 RTSS Live Streaming

- Ahead-of-Time segment requests
- Explicit discontinuity support + Fast Recovery
- Multiple Stream Synchronization (synchronize multiple camera angles)
- Same segment documents for RTSS and HLS/DASH
- SCTE-35 Advertising Signals
- Extensible Real-Time Signals
- Proxy-based server-side universal advertising insertion
- Negates interruptions due to TLS re/negotiation, Google Safe Browsing SHA matches
- Negates dependency of the playlist to continue playing the same quality rendition
- ABR bandwidth calculations
- Supports irregular segment durations
- Capable of sustained 300 millisecond Screen-to-Screen delay

RTSS corrects the disadvantages of other specifications by defining a universal payload format and server design which can be used to play streams in real-time without sacrificing any of the benefits of playlist-based live streaming.

5.11.2 Periscope LHLS

- Ahead-of-Time segment requests
- Requires playlist updates
- No discontinuity support
- No advertising signaling because markers are not known ahead of time

- No defined method of ABR bandwidth calculations

5.11.3 DASH Live Profile

- Just-in-Time segment requests using a coordinated clock
- Any function which requires a new coordinated period created in the manifest document drastically increases the overall stream delay and may introduce stream stall.
- Discontinuity requires new coordinated clock period.
- Advertising signaling requires new coordinated clock period.
- Player cannot recover target latency after temporary publisher interruptions (when Index-based naming is used)
- Cannot support irregular segment durations without new coordinated clock period.
- Subject to delays caused by load balancing, TLS renegotiation, and hitting Google Safe Browsing SHA hashes

5.11.4 DASH HTTP/2 Server Push

- Cannot be CDN accelerated without designing a custom CDN
- Requires Proprietary API for ABR and Starting H2 Push Session
- No intentional segment drop to maintain desired latency

5.11.5 WebSocket Push

- Cannot be CDN accelerated without designing a custom CDN
- Requires Proprietary API for ABR and Starting Playback
- Server makes all content delivery decisions per-Viewer
- No discontinuity support
- No SCTE-35 support

6 RTSS Application Model

6.1 Media Presentations

A Media Presentation consists of a plurality of media segments representing a single timeline. Media Presentations are analogous with Tracks when only one media codec is utilized.

Unlike [CMAF], multiple [ISOBMFF] Tracks MAY be present in a single segment payload. Where [CMAF] compliance is required, RTSS Media Presentations are CMAF Track Presentations.

6.1.1 Timeline Alignment after Discontinuity

The `EXT-X-PROGRAM-DATE-TIME` signal and [ISOBMFF] `baseMediaDecodeTime` attribute are used to establish a world clock from which the alignment of a discontinuous timeline can occur without needing extra information from the playlist document.

```
TimelineOffset = EXT-X-PROGRAM-DATE-TIME – baseMediaDecodeTime
TimelineOffsetAfterDiscontinuity = TimelineOffset – (EXT-X-PROGRAM-DATE-TIME –
baseMediaDecodeTime)
```

6.2 Media Switching Set

A Media Switching Set contains one or more Media Presentations. All Media Presentations within the same Media Switching Set MUST use the same media codecs to ensure seamless switching.

It is REQUIRED that segment N represents the same `baseMediaDecodeTime` across all Media Presentations in the same Media Switching Set. This allows for "blind" switching between the quality renditions when requests are made *ahead-of-time*.

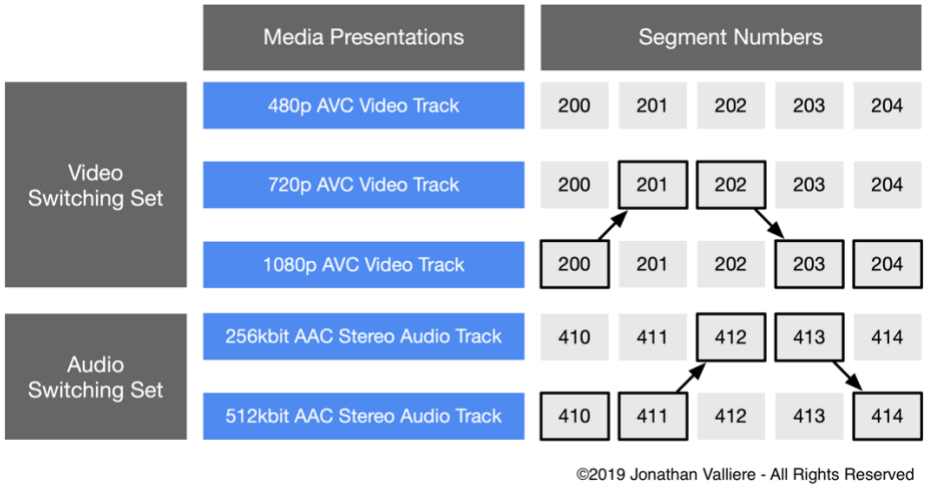

Figure N – Example Presentation Switching

This requirement follows guidelines set forth in [CMAF] Section 6.5.6.2

6.3 Media Segment Format

RTSS uses ahead-of-time segment requests to achieve low latency. In order to implement all of the playlist-specific features which would otherwise be lost, signaling is added into the segmentation payloads. Since segments are requested ahead-of-time, this signaling is essentially real-time allowing the stream to function identically to a stream based on a playlist with a fraction of the latency.

- Segments SHALL be coded in compliance with [ISOBMFF]. [CMAF] restrictions are OPTIONAL.
- Segments SHALL start with a [ISOBMFF] File Type Box (`'ftyp'`)
- Segments SHALL include at least one Real Time Signal Box (`'rtsb'`).
- Segments SHOULD have a duration no longer than 3 seconds. This requirement is intended to prevent false-positive long-polling timeouts.
- Segments SHOULD be finalized if more than 300 milliseconds elapses when no new data is appended.

6.3.1 RTSS Media Fragments

A single segment contains a plurality of [ISOBMFF] Movie Fragments. [ISOBMFF] Movie Fragments are a combination of a `` `moof` `` and `` `mdat` `` box. To prevent confusion, [ISOBMFF] Movie Fragments are identical as [CMAF] Chunks. For the purposes of RTSS, Movie Fragments SHOULD HAVE a maximum duration of 250ms. The minimum Movie Fragment duration SHOULD be large enough for effective ABR calculations, 100ms is RECOMMENDED.

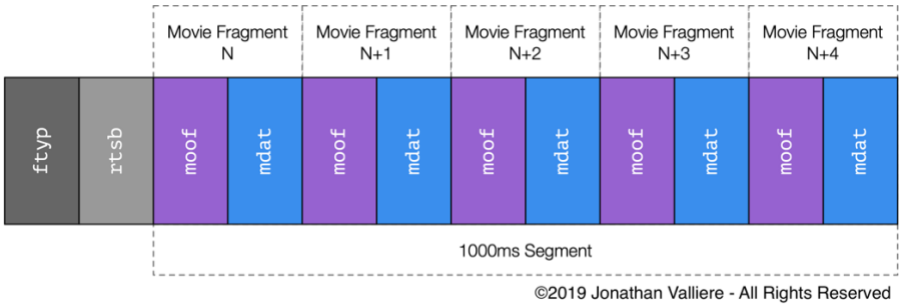

Figure O - Fragment Duration Example

6.3.1.1 Calculating Fragment Delivery Jitter for ABR (Adaptive Bitrate) decisions

Jitter calculations are used to determine the relativistic timeliness of fragment delivery. When specified thresholds are exceeded, quality must be dropped. Epoch calculations are processed once at the beginning of a stream or discontinuity.

```
CurrentTimeMills = 4800
BaseMediaDecodeTime = 90

StreamAlignmentEpoch = CurrentTimeMills - BaseMediaDecoeTime
StreamAlignmentEpoch = 4710

CurrentTimeMills = 4950
BaseMediaDecodeTime = 190

FragmentJitter = (StreamAlignmentEpoch + BaseMediaDecodeTime) - CurrentTimeMills
FragmentJitter = -50
```

In the example above, the Fragment arrived 50ms later than expected; when FragmentJitter is greater than 1/3 of the desired playback buffer, ABR downgrade is desired.

6.3.2 RTSS Signals

Signals are used to indicate unpredictable changes in stream behavior in real-time. This includes such operations as discontinuity, SCTE-35 advertising markers, stream termination, or codec changes.

6.3.2.1 Real Time Signal Box (`'rtsb'`)

The Real Time Signal Box (`rtsb`) is a top-level [ISOBMFF] object which provides a UTF-8 coded String representing M3U8 elements. Multiple `rtsb` boxes MAY be present at the beginning and at the end of a segment. Theses boxes MUST NOT be sandwiched between two Media Fragments (`moof`). The `rtsb` box MUST ONLY be parsed when intending to play the stream in compliance with this specification.

By including the signaling box in the segments, existing HLS playlist tags may be capitalized when using ahead-of-time segments.

When generating a backwards compatible stream, segments only containing the `rtsb` box MUST NOT be added to the playlist; this is likely to occur during end-of-stream, discontinuities, and lazy server-side ad insertion.

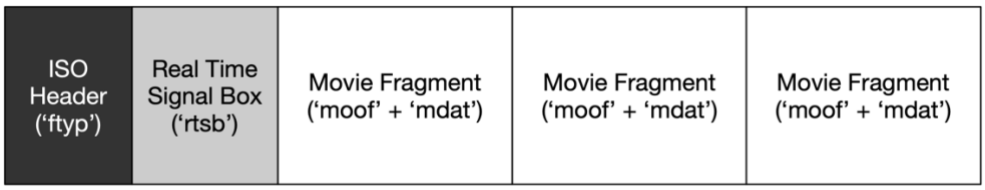

Figure P - Example Segment with Real Time Signal Box

6.3.2.2 Real Time Signal Definitions

There are no limitations of the type of signals used in the `rtsb` box. However, all signals must be compliant with [M3U8]. Below is the list of supported tags included from [HLS]. While it is possible to use additional tags described in [HLS], any tags not listed below are not supported as part of this standard.

EXT-X-PROGRAM-DATE-TIME	MUST be present, in all segments, before the first Media Fragment, in order to calculate track and stream synchronization except where EXT-X-DATERANGE is used. Additionally, this signal provides increased synchronization accuracy after discontinuities.
EXT-X-DISCONTINUITY-SEQUENCE	MUST be present, in all segments, before the first Media Fragment, after the first discontinuity.
EXT-X-DISCONTINUTY	MUST be present before the first Media Fragment when a timestamp discontinuity occurs.
EXT-X-ENDLIST	MUST be present in the last segment of a stream. Media Data MUST NOT be present.
EXT-X-DATERANGE	Used to signal SCTE-35; Replaces EXT-X-PROGRAM-DATE-TIME when present.
EXT-X-KEEPALIVE	Used to signal the stream is alive but media is not present.
EXT-X-MAP	The EXT-X-MAP tag specifies how to obtain the Media Initialization required to parse the applicable Media Segments. EXT-X-MAP must be included in every segment not containing the MOOV box.

In the future, Signals may be defined to control player behavior, such as preventing scrubbing or indicating interactive advertising information.

6.3.2.3 How Signals are Mapped in a Playlist

In **Figure Q**, an example is shown how signals can be used indicate gaps in media for the purpose of client-side advertising insertion. When using signals, it is always desirable finalize the current Media Segment than to have a period of time in which no new data is appended. This is desirable because Middleware systems may assume that long interruptions in data transfer are the result of an uncorrectable failures.

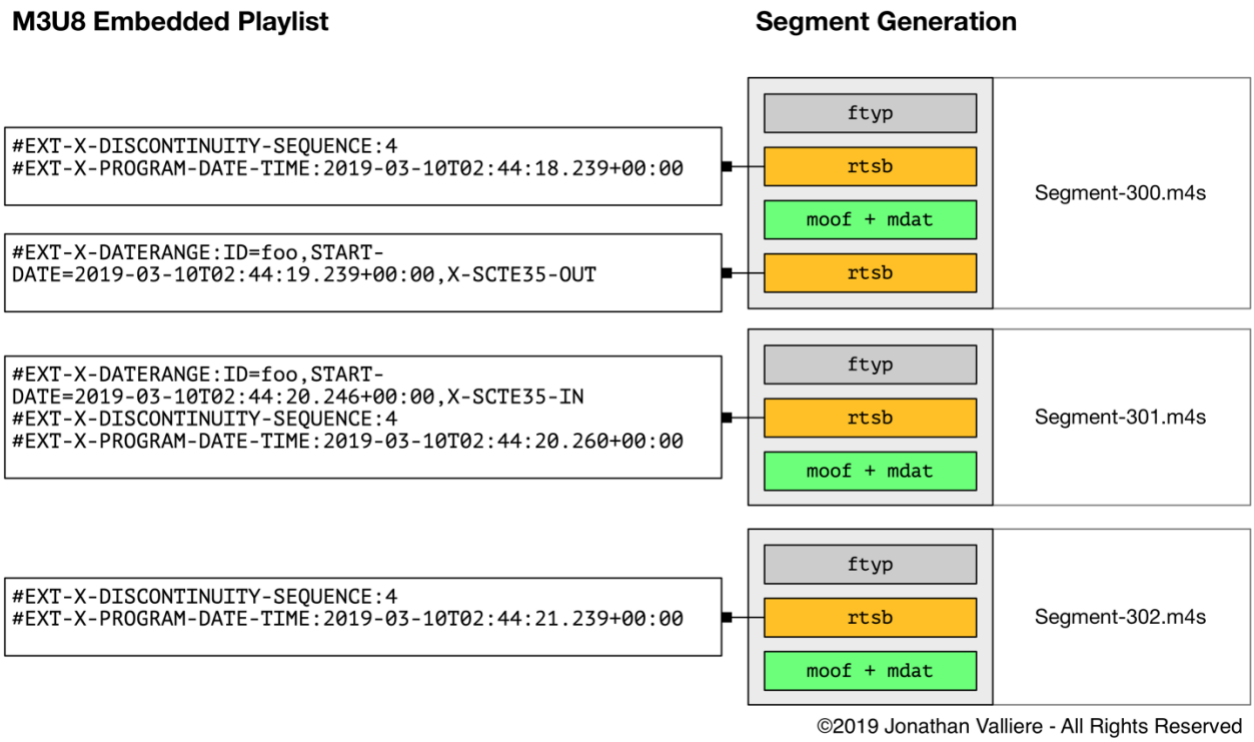

Figure Q - Example of Segment Playlist Mapping

7 RTSS Segment Encoder

The RTSS Segment Encoder, as shown in **Figure J**, is responsible for converting a broadcast of Media Frames into Segment documents and inserting RTSB signals, defined in **Section 6.3.2.1**, as required. A major benefit of RTSS is that specialized Media Capture + Media Encoder + Segment Encoder software is not required to achieve sustainable low-latency, unlike CMAF DASH-LIVE. Due to the real-time nature of the RTSS Segment Encoder, additional downstream RTSS Segment Encoders may re-encode the stream for the purpose of inserting advertisements without increasing the overall Playback Delay.

The RTSS Segment Encoder is also responsible for generating Manifest and Playlist documents, as desired, for compatibility with other streaming formats. Manifest and Playlist documents shall include methods of indicating compatibility with RTSS and provide the necessary configuration information. This specification does not currently suggest amendments for HLS and DASH to support RTSS.

The RTSS Segment Encoder SHOULD NOT produce segment data unless either a Media Fragment is available, or an Advertising Signal is required. For example, if the stream is currently away due to an advertisement period, the `ftyp` and other metadata must not be written to the segment. By not writing any data to the empty segment, the HTTP server can produce an “408 Request Timeout” message when appropriate. This helps Middleware components understand what is happening with the stream.

8 RTSS Server

8.1 General Requirements

- Server MUST be capable of suspending error messages for requests for segments which are anticipated to exist in the future. This allows the Player to request a segment ahead-of-time it exists and wait until it does.
- Server MUST track relative position of segment streams in order to determine whether a request for a specific segment was created in the past or *should* be created in the future. This allows for Players to navigate a number of tracking errors in order to maintain relative synchronization with the stream.
- Server SHOULD limit the maximum number of segments a Player may request ahead of the most recent known segment to exist. Usually, the maximum should be between 3-5.
- Server SHOULD support HTTP/2 to prevent simultaneous request limitations imposed upon by the User-Agent.

8.2 HTTP Requirements

- Requested segments that *should* come into existence after the most recent known segment MUST receive a HTTP "408 Request Timeout" message if the requested segment does not come into existence after a period of time.
- Requested segments too far in advance of the most recent known segment MUST receive HTTP "425 Too Early" response suggesting that the Player is requesting segments too far in advance.
- Requested segments that are older than the most recent known segment and have been deleted by the server MUST respond with HTTP "410 Gone".
- HTTP "404 Not Found" responses are ONLY allowed after the Stream has terminated.

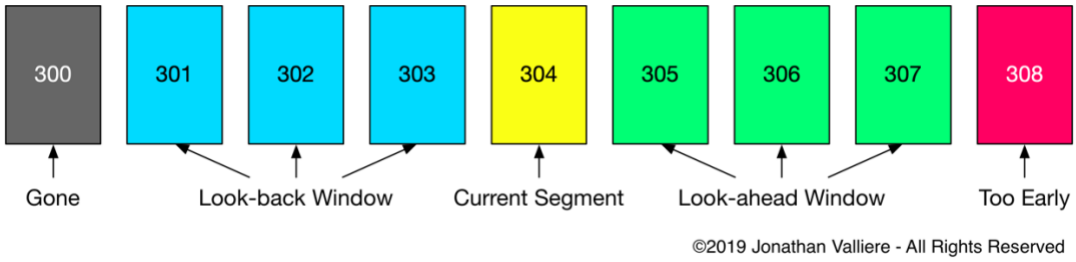

Figure R - Server Relative Segment Tracking

8.3 URL Structure for Streams (Optional)

A predictable structure comes with significant advantages concerning optimizations of the delivery workflow, stream transformations, and the creation of new low latency monetization options. This URL Structure is used for Publishing and Middleware components. URL transformations are allowed by downstream components.

```
{base-path}/{stream-id}/{presentation-id}/{segment-number}
```

8.3.1 General Requirements

- Each Stream MUST expose "manifest.m3u8" document complying with the HLS manifest format inside of the base stream URL.

- Each Presentation MUST expose “playlist.m3u8” document complying with the HLS playlist format inside the base presentation URL.
- Segment Numbers MUST rollover after a maximum value of a 32-bit unsigned integer.
- Segment Names MUST follow the following format “media-%Number%.m4s”

8.3.2 Benefits of URL Structure

- Stream can be reverse-navigated by a middleware system for business specific purposes such as server-side advertisement insertion or intelligent pre-caching. e.g. by having a URL of a segment, the server can locate the manifest document listing all Media Presentations then pre-fetch segments belonging to the same Seamless Switching Set.
- Concurrent Connections and Task Scheduling of middleware systems can be optimized when RTSS Streams are detected versus other traffic types.
- Low Latency Live Segments can be easily processed into DVR streams in real-time.
- URL hashing is more optimized which helps storing related content on the same disk.
- Simplifies server log processing for per-presentation usage analytics.
- Simple out of order and hole detection using sequential segment numbers.

9 RTSS Player

9.1 General Operation

URLs for Segments MUST be generated using a predictable pattern or MUST be supplied to the Player significantly in advance of their availability as to protect against delays caused by downloading an updated playlist through a content delivery network. How this occurs it outside of this specification.

In order to maintain real-time latency, two segments from every active presentation MUST be requested at all times by the Player as described in **Section 5.6**. This is depicted by **Figure S** below. In order to better visualize multiple concurrent requests, the flow is broken into 2 separate columns.

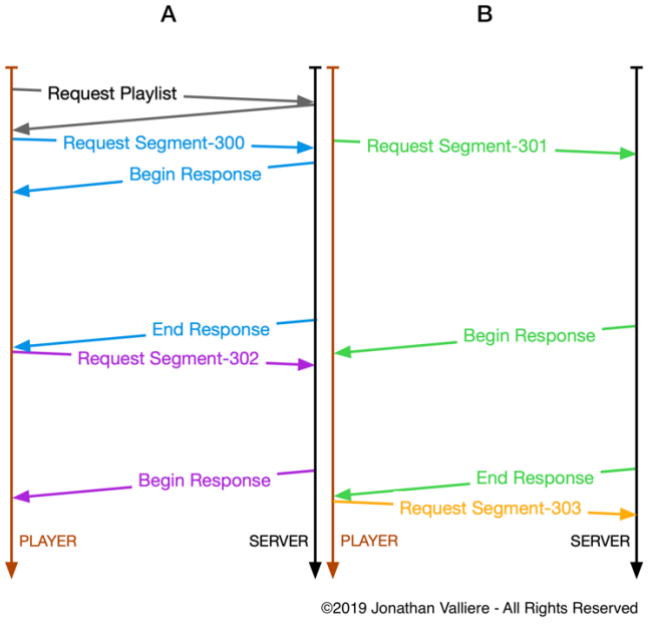

Figure S - Player Request Flow Diagram

This requirement allows for segments to be arbitrarily terminated without causing a delay in receiving data for the next segment. For example, as response for segment-300 completes, the next request for segment-302 is sent as shown in **Figure S**. By automatically moving forward from one segment to the next, the Player can align itself with the segment creation cadence of the server by searching for the first segment which IS NOT delivered faster than real-time.

In order for the Player to determine that media is being delivered in real-time, only one HTTP Request must be receiving data. Where two cascading HTTP Requests are being utilized per Presentation, only one must be receiving data otherwise real-time is not established.

Note: Use of HTTP/2 is RECOMMENDED to prevent negative effects of browser-imposed connection limits. When possible, it is RECOMMENDED to use multi-track presentations to reduce the number of concurrent requests. For example, subtitles can be easily combined with audio.

Segment data is received, in real-time, and appended to the Player Buffer. How much buffer the Player requires is outside of this specification.

9.2 Player Requirements

- Player MUST limit the number of ahead-of-time segment requests to two per Media Presentation.
- Player MUST employ a connectivity timeout mechanism to detect segment response stalls. Segment Encoder has an option to produce Segments with `EXT-X-KEEPALIVE` signals to prevent this timeout from occurring.
- Player MUST implement a back-off strategy when polling for a Segment from the Web Server which is returning HTTP “404 NOT FOUND” messages.
- Player MUST update playlist and attempt to resynchronize / restart stream after receiving HTTP “425 Too Early” response from the Web Server.

10 RTSS Advertising Insertion

Stream monetization is a core requirement for any streaming protocol. However, the desire for lower latencies introduces certain complications into established methods of inserting and signaling advertisements without increasing the stream delay.

10.1 Server-Side Ad Insertion

Server-side Ad Insertion is often sought to prevent ad-blockers from preventing advertisements from functioning correctly. RTSS, being a real-time protocol, is capable of delivering media to ad-insertion servers without increasing the end-user stream delay. When advertising is inserted into the stream, the segment numbers must be remapped accordingly as shown in **Figure T**.

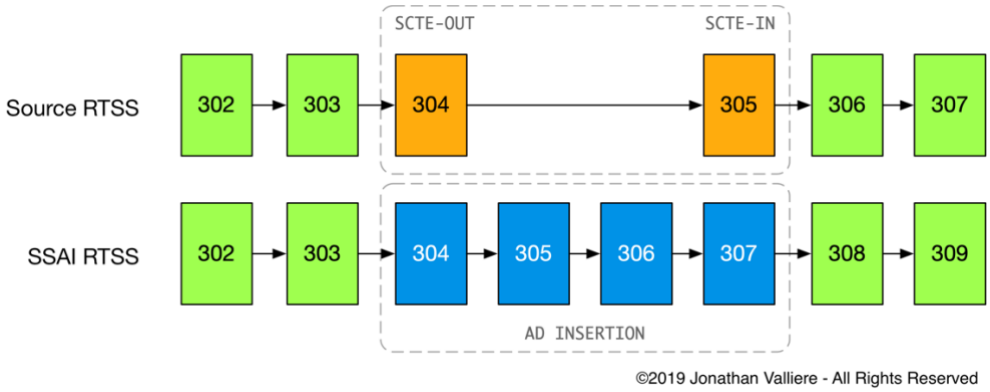

Figure T - Server-Side Ad Insertion Example

10.2 Client-Side Ad Insertion

Client-Side Ad Insertion occurs when the client receives the appropriate SCTE signals. Advertisements must be pre-loaded to prevent delay.