Introduction

This is an attempt to document the issues that Twitch/Amazon IVS has encountered with various distribution protocols over the last 8 years.

HLS

We initially used RTMP for distribution but switched to HLS something like 8 years ago. We use MSE to feed the player buffer on web platforms. This assumes 2s segments.

Congestion

- Must finish downloading the current segment before ABR can switch renditions
- ABR cannot differentiate between broadcaster starvation and network congestion
- Depleting the buffer causes a jarring pause while it refills

Latency

- Segments must be finished before they are added to the playlist, adding 2s
- Playlist updates must be polled by the player, adding up to 2s
- Segments are downloaded sequentially, adding 200ms? between requests
- MSE has a minimum buffer size, adding 100ms?
- BBRv1 introduces latency during the PROBE_RTT phase, adding 400ms?
- Depleting the buffer increases latency for the remainder of the session

Time to Video

- Fetching the first rendition playlist takes 3 RTTs.
- Fetching the first segment takes 3 RTTs.
- The buffer needs to be sufficiently filled before playback starts.

Clients

- Our server can't do anything to improve dumb clients
- Limited 3rd party metrics

LHLS

We made our own fork of HLS to address some of the issues mentioned above. Segments are advertised in the playlist ahead of time and delivered frame-by-frame with HTTP chunked-transfer.

AS COMPARED TO **HLS**

Congestion

- **(new)** Individual frames are often not large enough to saturate a network.
- **(new)** ABR has difficulty determining when it can switch up

Latency

- ~~Segments must be finished before they can be downloaded, adding 2-4s~~
- ~~Playlist updates must be polled by the player, adding 0-2s~~
- **(new)** We benchmark the network between segments, adding 200ms?

Clients

- ~~Our server can't do anything to improve dumb clients~~
- ~~Limited 3rd party metrics~~
- **(new)** 3rd party clients are not supported

Performance

- **(new)** Frame delivery is more expensive due to context switching

LL-HLS

Apple went ahead and made their own low latency HLS solution. Segments are split into sub-segments and updates are requested more frequently. We have not implemented this yet so some of these bullet points may be inaccurate or missing. This assumes 2s segments and 500ms sub-segments

AS COMPARED TO **LHLS**

Congestion

- ~~Individual frames are often not large enough to saturate a network.~~
- ~~ABR has difficulty determining when it can switch up~~

Latency

- ~~A blocking speed test between segments is required to saturate the network~~
- **(new)** Sub-segments must be finished before they are added to the playlist, adding 500ms
- **(new)** Playlist updates need to be pushed to the player, adding 100ms?

Clients

- ~~3rd party clients are not supported~~
- **(back)** Our server can't do anything to improve dumb clients
- **(back)** Limited 3rd party metrics

Performance

- ~~Frame delivery is more expensive due to context switching~~
- **(new)** 4x the number of playlist and segment requests
- **(new)** Playlist long-polling involves more context switching

WebRTC

We decided that the only way to further reduce latency was to use WebRTC. This project involved using WebRTC for last-mile delivery; our existing video system (RTMP ingest, HLS distribution) was used for everything prior. We tried twice; once using libwebrtc and another time using a heavily forked pion.

Some of these issues would not be present if we replaced our entire video pipeline with WebRTC instead of this hybrid approach. That would have been a gigantic undertaking and was absolutely not feasible.

AS COMPARED TO **LHLS**

Congestion

- ~~ABR has difficulty determining when it can switch up~~
- ~~No way to differentiate between broadcaster starvation and network congestion~~
- ~~Individual frames are often not large enough to saturate a network.~~
- ~~Depleting the buffer causes a jarring pause while it refills~~
- ~~Adds latency for the remainder of the session~~
- **(modified)** Depleting the jitter buffer causes frame dropping
- **(new)** Dropping reference frames causes artifacts or freezing until the next IDR
- **(new)** RTCP receiver report is awful for congestion control; required transport wide CC instead

Latency

- ~~Segments are downloaded sequentially, adding 100ms between requests~~
- ~~We saturate the network between segments, adding 200ms~~
- ~~BBR causes latency during PROBE_RTT phase~~
- ~~Depleting the buffer increases latency for the remainder of the session~~
- **(new)** An additional jitter buffer is required in front of WebRTC, adding 100ms?

Quality

- **(new)** H.264 B-frames are not supported, causing a VQ loss
- **(new)** Producing fewer reference frames causes a VQ loss
- **(new)** Excess jitter in the video pipeline (ex. ingest) causes playback jitter
- **(new)** Any transcoding changes also impacts our HLS stack (VQ loss)

Time to Video

- ~~Fetching the first rendition playlist takes 2-3 RTTs.~~
- ~~Fetching the first segment takes 2-3 RTTs.~~

- ~~The buffer needs to be sufficiently filled before playback starts.~~
- **(new)** Negotiating SDP via HTTPS takes 3 RTTs.
- **(new)** Negotiating ICE takes 2-3? RTTs
- **(new)** Negotiating DTLS takes 2? RTTs

Clients

- **(new)** libwebrtc is obnoxious to build
- **(new)** libwebrtc bloats the size of our player
- **(new)** Limited user experience metrics
- **(new)** Transport wide CC extension was not supported by all clients/browsers

Features

- **(new)** Does not support DRM

Performance

- **(new)** UDP delivery is more expensive than TCP in practice
- **(new)** Requires transcoding to remove B-frames
- **(new)** Requires transcoding to convert AAC to OPUS
- **(new)** libwebrtc did not scale to hundreds of viewers, let alone thousands

Frames over WebRTC data channels

When WebRTC was not working, we tried to switch over to WebRTC data channels (SCTP over DTLS). Each frame was sent as a WebRTC data channel message. These frames could be fed into the player via MSE.

It didn’t work. SCTP deadlocks when messages are too large because they count towards flow control until fully received. The flow control limits in Chrome and Firefox are hard-coded and are often smaller than a single I-frame. SCTP cannot drop messages out of order.

(continued on next page)

RTP over WebRTC data(grams)

Since data channels weren't working as intended, we decided to send each RTP packet as an unreliable message. This was then reassembled by the application and fed into the player.

AS COMPARED TO **LHLS**

Congestion

- ~~ABR has difficulty determining when it can switch up~~
- **(new)** SCTP has poor congestion control (I forget why)

Latency

- ~~We saturate the network between segments, adding 200ms~~
- ~~Segments are downloaded sequentially, adding 200ms? between requests~~

Time to Video

- ~~Fetching the first playlist takes 2-3 RTTs.~~
- ~~Fetching the first segment takes 2-3 RTTs.~~
- **(new)** Negotiating SDP via HTTPS takes 3 RTTs.
- **(new)** Negotiating ICE takes 2-3? RTTs
- **(new)** Negotiating DTLS takes 2? RTTs
- **(new)** Negotiating SCTP takes 2? RTTs
- **(new)** Negotiating data channels takes 1 RTT

Features

- **(new)** Does not support 3rd party CDNs

Performance

- **(new)** UDP delivery is more expensive than TCP in practice
- **(new)** SCTP ACKs cause excessive UDP packets to be sent/received

Warp

Warp is conceptually similar to LHLS, but segments are pushed in parallel via QUIC/WebTransport. Prioritization is used to avoid segments fighting for bandwidth, delivering newer media first (especially audio) during congestion.

AS COMPARED TO **LHLS**

Congestion

- ~~ABR has difficulty determining when it can switch up~~
- ~~Must finish downloading the current segment before ABR can switch renditions~~
- ~~No way to differentiate between broadcaster starvation and network congestion~~
- ~~Individual frames are often not large enough to saturate a network.~~
- ~~Depleting the buffer causes a jarring pause while it refills~~
- **(modified)** Depleting the audio buffer causes a jarring pause while it refills
- **(new)** Depleting the video buffer causes frames to be skipped
- **(new)** PING packets must be sent to occasionally saturate the network

Latency

- ~~We benchmark the network between segments, adding 200ms?~~
- ~~Segments are downloaded sequentially, adding 200ms? between requests~~
- ~~Depleting the buffer increases latency for the remainder of the session~~
- **(modified)** Depleting the audio buffer increases latency for the remainder of the session

Time to Video

- ~~Fetching the first rendition playlist takes 3 RTTs.~~
- ~~Fetching the first segment takes 3 RTTs.~~
- **(new)** WebTransport handshake takes 2 RTTs

Clients

- **(new)** Chrome only WebTransport support
- **(new)** Chrome only video underflow support

Features

- **(new)** Does not support 3rd party CDNs

Performance

- **(new)** UDP delivery is more expensive than TCP in practice