IMPROVING LIVE PERFORMANCE IN HTTP ADAPTIVE STREAMING SYSTEMS

Kevin Streeter

Adobe Systems, USA

ABSTRACT

While HTTP adaptive streaming (HAS) technology has been very successful, it also generally introduces a significant amount of live delay as experienced by the end viewer. Multiple elements in the video preparation and delivery chain contribute to live delay, and many of these elements are unique to HAS systems versus traditional streaming systems such as RTSP and RTMP. This paper describes how improvements both in the structure of the media, the delivery workflow, and the media player can be combined to produce a system that compares well with broadcast. The paper concludes with a preview of advances in delivery technology (such as HTTP2) that will improve the experience even more in the near future.

INTRODUCTION

While HTTP adaptive streaming (HAS) technology has been very successful in delivering stable over-the-top video experiences at large scale, the technology has a number of important limitations as well. One significant limitation is the introduction of large delays for live content, presenting the content to the viewer far later than in traditional broadcast systems. This issue makes it very difficult for today's broadcasters to provide over-the-top digital services for sports and other live events with quality-of-experience that compares well to earlier broadcast systems. Working around these limitations is challenging due to the fact that the delays are not caused by a single component, but are introduced throughout the delivery system.

This paper provides a detailed analysis of the mechanisms of HTTP adaptive streaming systems that lead to this delay, showing how each element in the delivery system contributes to the problem. The paper then describes how improvements both in the structure of the media, the delivery workflow, and the media player can be combined to produce a system that compares well with broadcast. The paper concludes with a preview of advances in delivery technology (such as HTTP2) that will improve the experience even more in the near future.

PREVIOUS WORK

This paper builds on earlier work by Swaminathan et al (1) and by Bouzakaria, et al (2) which describe the use of reduced size delivery units and HTTP chunked transfer coding to eliminate delay in HAS systems caused by the need to accumulate at least one

complete segment of media before that media can be transferred to a downstream processor. These works note that this behavior introduces delay of at least the duration of the segment (which is commonly 4s-10s long), plus any additional delay associated with connection setup time, etc.

To address the impact of segment size, these systems break up individual media segments into smaller duration units ("chunks"), which can be incrementally transferred using HTTP's "chunked transfer encoding" capability. This allows the transfer to begin before the entire segment has been accumulated. Live delay then becomes a function of chunk duration, rather than of segment duration.

While this method is effective in reducing live delay introduced by segment duration, it does have some drawbacks. This technique relies on specialized behavior by the media source and origin server to begin sending chunked data before the segment is complete. If either element is not aware of the need to send data as soon as it becomes available, it will negate the reduction in live delay. In addition, standard HTTP cache semantics do not describe how a caching proxy should behave when receiving chunked content, and in most cases a standard proxy would not retransmit the chunked media until the entire segment had arrived. Again, this reintroduces live delay into the system.

In this paper, we will extend the architecture described in this previous work to include a multi-hop HTTP caching layer that is consistent with the topology of many Content Delivery Networks (CDN) operating today. We will also describe techniques that utilize new capabilities in HTTP to decrease live delay without introducing specialized behaviors at the HTTP layer.

DEFINING LIVE DELAY

For the purposes of this paper, we will define live delay simply as the amount of wall clock time that has passed between when an event has occurred "in real life" (for example, a goal is scored at a football match) and when that same event can be seen on the screen of a media playback device by a person. In this definition, details of what happens to get the media presentation from the location of the event to the viewer are not important.

We can contrast this with another common measure of delay, which is "start-up delay". This measure is the amount of wall clock time between when a user of media playback devices presses "play" on the device (or switch channels, or other way method of initiating playback) and when the media presentation first appears. Note that while live delay and startup delay may be related (for example, filling a large buffer in the playback device will generally increase both measures) they have different causes and a different impact on the experience of the end-user. This paper will focus on the reduction of live delay, rather than of start-up time.

CONTRIBUTION TO DELAY IN MULTI-HOP HAS SYSTEMS

We can describe the contributions to delay in a multi-hop HAS delivery system as follows:

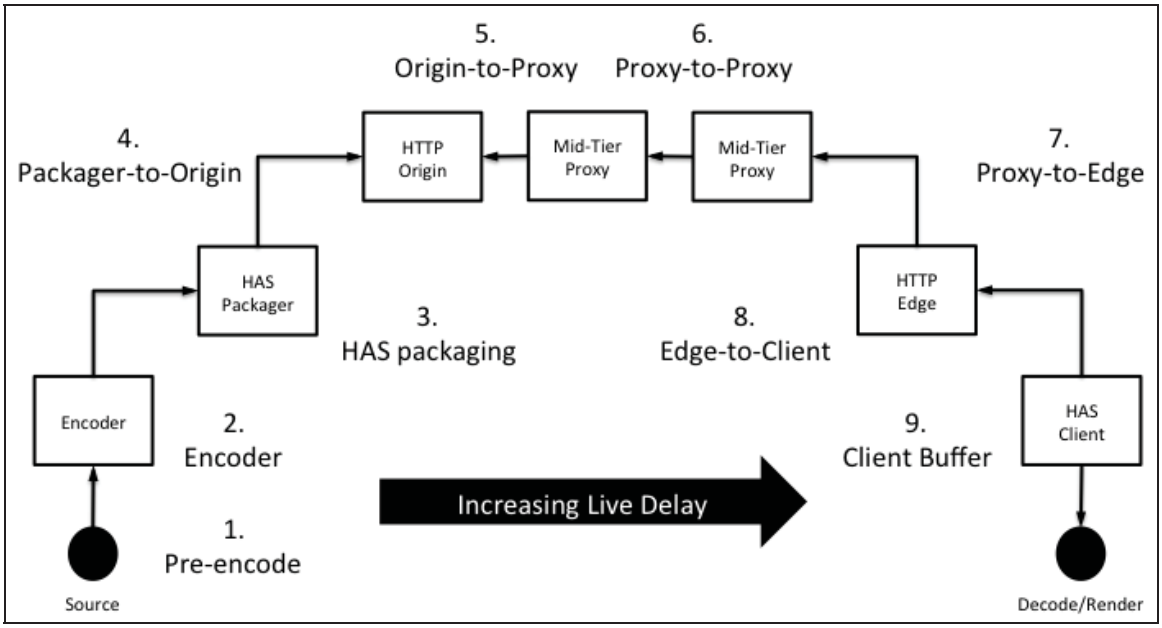

Figure 1 - Contribution to Live Delay

Referring to Figure 1, we describe nine sources of live delay in the system:

1. Pre-encode Delay. The pre-encode delay is the time between the image entering the capture device (for example, a video camera) to when a representation of that image is available at the encoder. This may include image processing at the capture device, and link transmission time between the capture device and the encoder.
2. Encoding Delay. The encoding delay introduced by the encoder. This delay is introduced by the encoder to support the video coding process, either by implementing look-ahead to increase coding efficiency or by delaying frames to implement bi-directional prediction (b-frame) intercoding.
3. HAS packaging delay. The HAS packaging delay is the time required by the encoder/packager to accumulate a single HAS segment, before that segment can be delivered upstream to an HTTP network ingest point (the HTTP origin).
4. Packager-to-Origin Link Delay. This is the time required to transmit a completed HAS segment from the packer to the origin.
5. Origin-to-Proxy Delay. This is the amount of time required to transmit a completed HAS segment from the origin to any intermediate, caching proxy nodes. This value may be zero on a cache “hit”, where the segment is already available at the proxy.
6. Proxy-to-Proxy Delay. If the HTTP network has multiple intermediate proxy tiers, delay will be incurred at each hop.
7. Proxy-to-Edge. This is the delay incurred transmitting a media segment between the final (internal) proxy tier and the edge of the HTTP network.
8. Edge-to-Client. This is the delay incurred transmitting a media segment from the network edge to the media client
9. Client Buffer. This is the amount of time required to accumulate a sufficient playback buffer at the client to decode video, manage network jitter, and manage adaptive switching logic.

AVERAGE DELAY, PEAK DELAY, AND VARIABILITY

An important observation about the architecture described above is that while some contributions to delay are constant (for example, the encoding delay), many contributions vary dynamically. Specifically, link transmission times will vary based on network conditions and effective bandwidth, and the delay occurred between hops (between origin, proxy, and edge tiers) will vary depending on both link transmission times *and* whether the media segment is available in the proxy's cache. This creates a high degree of variability, which can only reasonably be modeled by calculating the peak delay incurred, assuming all caches "miss".

This behavior means that the average delay incurred may be much lower than the peak delay, but variability in the system forces a media client to view the system as if it operated at peak delay. For this reason, to reduce the effective delay in the system we need to not only reduce average latency, but also limit delay variability.

EFFECT OF SEGMENT SIZE ON DELAY

It can be noted that chosen duration of the media segment has a direct impact on the live delay, at step 3 in the diagram. If a media segment is on average D seconds in duration, the HAS packaging delay will on average be at least D seconds. Reducing the size of the segment therefore directly reduces live (peak) live delay in the system.

While reducing segment size is an effective way of reducing live delay, there are limitations to how small the segment can be made. In his paper on the subject, Thornburgh (3) shows the impact that TCP congestion control and slow start have on the transmission of very small segments. This places a floor on how small a segment can be, unless some other mechanism is used to compensate.

EFFECT OF HTTP "PULL" MODEL ON DELAY

To understand some of the limitations of HAS, it is useful to compare it to traditional streaming technologies such as RTSP and RTMP. In a traditional streaming system, clients maintain a connection to the media server, which transmits media via this connection. For streaming over the Internet, this connection is generally made via TCP, and is unicast. In some cases the connection may be handled via UDP, and may be multicast. In any of these cases, the media transmission is paced according to the timing of the media presentation. These systems can therefore be described as "push" based systems.

Unlike HAS, traditional streaming utilizes specialized media servers that are aware of the internal details of the media content. This allows the server to manage system resources like disk and network utilization, in such a way as to allocate only the resources required to maintain correct stream pacing. Similar to HAS, media servers may be organized into multiple tiers, forming what is known as an edge/origin architecture.

The use of specialized servers typically allows streaming systems to deliver data with very low latency. RTMP and RTSP live services frequently deliver streams over the Internet at scale with end to end latency of 2 to 3 seconds, most of which is caused by buffers maintained on the client. Compare this with typical HAS systems such as Apple HLS,

which often deliver live services with end to end latency from 20 seconds up to 60 seconds or more.

The limitations of today's HTTP streaming systems can in general be traced to the request-response "pull" mechanics used by the client to access the media. As can be seen in the architecture described above, this behavior can lead to high variability in respect to when the media is available for the client to fetch. This variability in when content becomes available forces the client to use large playback buffers in order to compensate. In addition, as mentioned earlier source variability is additive to the throughput variability introduced by network conditions. This means that a client must allow for enough playback buffer to account for both effects combined (as opposed to which effect is greater).

IMPROVING LIVE DELAY USING "PUSH"

We can reduce or eliminate the impact of variability caused by the HTTP "push" model by introducing "push" behavior similar to that of the traditional streaming systems described above.

There are two main elements to such a system:

- Media-aware nodes. Unlike "dumb" HTTP infrastructure, that don't understand media in any way, push based systems understand the temporal nature of media. The system uses this understanding to ensure that downstream delivery of media is paced in a way consistent with the media timeline. This reduces live delay by eliminating the cache-layer variability and leaving only the transmission variability.
- Push-based transmission. In order for media aware processors to transfer media at a rate consistent with the media timing, they need to control pacing of the transmission. In other words, they need a way to push media data onto the link as needed.

With these two elements, it is possible to significantly reduce the live delay at each point in the transmission path.

USING HTTP/2 "PUSH" IN HAS SYSTEMS

There are a number of ways outside of HTTP to implement push-based transmission. For example, a system might be implemented to use RTSP, RTMP, or multicast methods to stream data from encoder to the edge of the network. While this would improve live delay, it would have the disadvantage of forcing the implementer to manage the transition of HAS media into and out of the non-HTTP elements of the system.

One method under active research is to enable push streaming without stepping outside the HTTP model, using the new "server push" capability of HTTP/2. This capability allows for the server to push any number of additional objects to the requesting client, in addition to the object being requested. There are limitations to this model (for example, all objects to be pushed in a single transaction must be "promised" before the original request can be serviced), but pushed objects are managed completely by the HTTP infrastructure and

remain the same HTTP semantics (such as cache-ability) as any other HTTP object. See Sheng et al (4) for a more detailed description.

When using this capability for streaming HAS media, the overall impact is to reduce delay variability across the HTTP network, which as noted above reduces the peak delay incurred across the system. Using HTTP/2 push, you can achieve live delay performance that is closer to that of traditional streaming systems, while retaining all of the advantages of the open HTTP architecture.

In addition to reducing delay variability, it is also possible to use HTTP/2 push to reduce delay by sending initialization information or other data required to quickly begin playback when the HAS manifest (for example, an MPEG-DASH MPD or HLS M3U8) is first requested. While this has the most impact on improving start-up delay, it also has the benefit of removing a TCP round-trip for this operation, which improves Edge-to-Client delay by an equivalent amount. See Cherif et al (5) for more on using HTTP/2 for "fast start".

CONCLUSIONS

This paper has described how the "pull" nature of the HTTP model introduces live delay variability and thereby increases peak live delay in HAS systems. As media data already consumes more bandwidth as any other type of data over today's HTTP-based CDN infrastructure, and this usage is only growing, it is reasonable to assume that the HTTP infrastructure will become more media aware.

Using techniques such as HTTP/2 server push, media aware infrastructure has the potential to improve streaming performance, such as by decreasing live delay in the ways described here. With the high-level of active research on this area, we can hope to see these techniques being used to improve production streaming delivery systems soon.

REFERENCES

1. Swaminathan, V.; Wei, S., "Low latency live video streaming using HTTP chunked encoding," Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on , vol., no., pp.1,6, 17-19 Oct. 2011

2. Bouzakaria, N.; Concolato, C. ; Le Feuvre, J., "Overhead and performance of low latency live streaming using MPEG-DASH," Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on. pp 92 – 97.

3. Thornburgh, M., "Calculating a Minimum Playable Duration for HTTP Streaming Media Segments," 2015 IEEE International Symposium on Multimedia (ISM). pp 519 – 522.

4. Wei, S.; Swaminathan, V., "Low Latency Live Video Streaming over HTTP 2.0," NOSSDAV '14 Proceedings of Network and Operating System Support on Digital Audio and Video Workshop. pp 37.

5. Cherif, W.; Fablet, Y.; Nassor, E.; Taquet, J.; Fujimori Y., " DASH fast start using HTTP/2," NOSSDAV '14 Proceedings of Network and Operating System Support on Digital Audio and Video Workshop. pp 25-30.