SCTE·ISBE CABLE-TEC EXPO 2018 ATLANTA, GA OCTOBER 22-25

2018 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Designing Video Services for Low-Latency Distributions in IPTV Cable Systems

A Technical Paper prepared for SCTE•ISBE by

Yasser Syed
Comcast Distinguished Engineer
TPX/VIDEO/VAST Dept., Comcast Cable
1701 JFK Boulevard Philadelphia, PA 19103 USA
303-246-8413
yasser_syed@comcast.com

Ali C. Begen, Professor, Ozyegin University, Turkey

Alex Giladi, Comcast Distinguished Engineer, TPX/VIDEO/VAST Dept., Comcast Cable

Table of Contents

List of Figures

Introduction

Service providers are moving towards IPTV (Internet Protocol Television) technologies using fragmented delivery as a way distribute video services over bandwidth-varying environments. This has already been successful with on-demand or pre-packaged video services (television shows, movies) and has been adapted, to some degree, for live event-based services (sports, news) -- but fragmented delivery works best for content that is already pre-packaged into a final format before delivery. Distributing live events, by contrast, presents additional latency-related challenges due to the real-time handling of requests, storage and delivery. For instance, customers expect to be able to record live broadcasts for later viewing. However, creating low-latency, live streaming by altering existing technology tools and structures simultaneously creates new and often prohibitive complexities. This paper examines some of the existing technologies in use today to optimize for network latency, and introduces possible implementations for achieving low-latency streaming.

1. Types of Latency

Latency can refer to one or more of latency causes in multimedia streaming that really depend on the application context. In live streams, latency is mostly discussed as an end-to-end latency, which describes the time delay between the action occurring in front of the camera and the same action being observed on the display. This is also known as glass-to-glass or handwaving latency. This cumulative latency can be broken up into three sub-parts that comprise of latency due to content preparation, latency for distributing it, and latency on the player.

For live events, content preparation latency is mostly associated with how the content is captured and encoded (including conditioning) through real-time processing using codecs such as AVC (Advanced Video Coding)/H.264 or HEVC (High Efficiency Video Coding)/H.265. Latency is inherently linked to the encoding process (e.g. look ahead) and the encoding structure through the use of temporal compression techniques that alter picture reordering in the coded stream, in order to increase picture quality while allowing for random access of the stream. Another part of content preparation latency is the packaging latency, which happens as the segment is received at all (or most) representations and parsed for indexing, so as to make it fetchable (manifests/indexing byte offsets) or streamable (multicast/uncast).

Once the content is prepared, it needs to be distributed. In file-based streams, distribution latency becomes one of the major factors discussed in this context. It is comprised of segment distribution (or the lack of it) to the CDN (Content Delivery Network) latency (CDNL), which does include delay for the I/O (input/output) storage of the segment. Another major factor is server latency, which describes the delay from the segment's availability on the server and the actual request for the content, to the receipt of it by the client player. This is often described as the delay from "the live edge". This type of latency is important for any type of file-based streaming, including pre-packaged content, because it prevents lag in player controls (e.g., seek, skip, rewind, fast forward). For live events, reducing this latency reduces the overall "glass-to-glass" latency.

The last sub-part to end-to-end latency is the player latency, which is a combination of the decode latency, buffer management and stream delay. Decode latency occurs relative to the coding structures used by the codec. Buffer management uses latency to avoid buffer underflow by extending the buffer to accommodate for "chunky" loading, due to the segment length, which may slow down the fill rate into the buffer. Lastly, stream delay is partly caused by player startup delay, which provides time to initially grow a buffer's capacity before playout occurs. This delay can avoid buffer underflow situations attributable to network

congestion that can slow down the buffer's fill rate. The startup delay is different from latency, which could take 5 seconds from pressing play, but the stream is only behind 3 seconds from the "live edge" (and the transmission time of the segment from the server.) The other part of stream delay is related to seeking behavior on the stream, and the time to initiate the action and see the result, which is a measure of delay when the end user time-shifts in the buffer.

Yet another type of latency happens at the system control plane level, where the number of requests to the server necessarily increases as a function of the number of players it handles. Above a certain capacity, this slows down the HTTP response time to send out segments to specific players, ultimately the filling rate into the client buffer. An increase in the number of players that a server handles would start making this type of delay more noticeable.

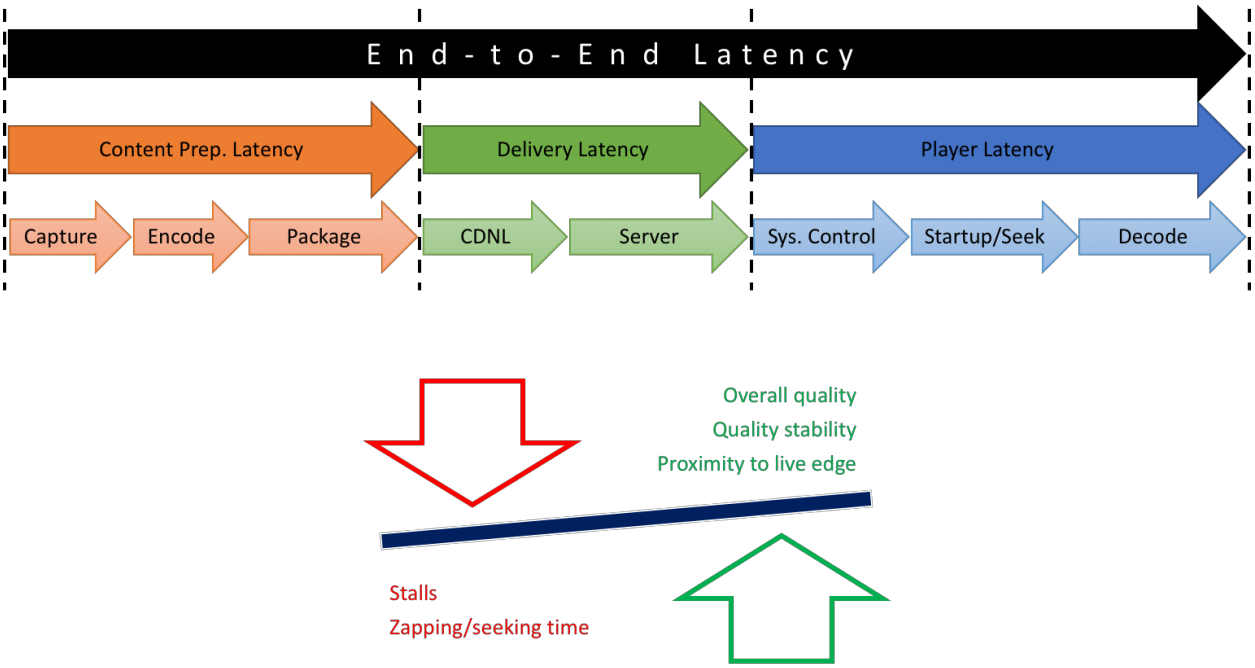

Figure 1: Types of latency in the end-to-end system.

A certain amount of latency is useful in network media streaming applications to ensure proper playback of the stream. It is a hedge against network jitter, keeps the player buffer filled, and allows for increased quality in low bitrate coding. Providing for low-latency is always a tradeoff against robustness of playback of the stream captured, and yet having low-latency streaming is important for live streaming. Additionally, developing low-latency techniques will also be helpful in emerging and latency-averse applications including, but not limited to, augmented and virtual reality (among many others.)

2. The Player/Decoder Model

In both live event and pre-packaged delivery, the player/decoder model is common. It is important to understand the mechanics of the player/decoder model and how it behaves, with respect to latency. Figure 2 illustrates a typical player/decoder model. The client makes requests for content segments from the HTTP server, which responds with the specified content segment for each request. This process starts filling the client buffer at a fast pace until the designated startup delay is achieved. Then, the client buffer starts to offload segments into the Coded Picture Buffer (CPB) at a rate to maintain the HRD (Hypothetical

Reference Decoder) model (an equivalent is done on the audio decoder side for audio segments), while simultaneously requesting new segments from the server. The goal is to maintain a constant client buffer level, which is much larger than the buffer maintained for the HRD model. When it encounters network congestion, the client player adjusts its requests according to a player algorithm to attempt to maintain a sufficient client buffer level. If the client buffer drains, then a rebuffering instance (observable as a stall in stream playout) happens. Other approaches may also attempt to adjust the emptying rate of the client buffer.

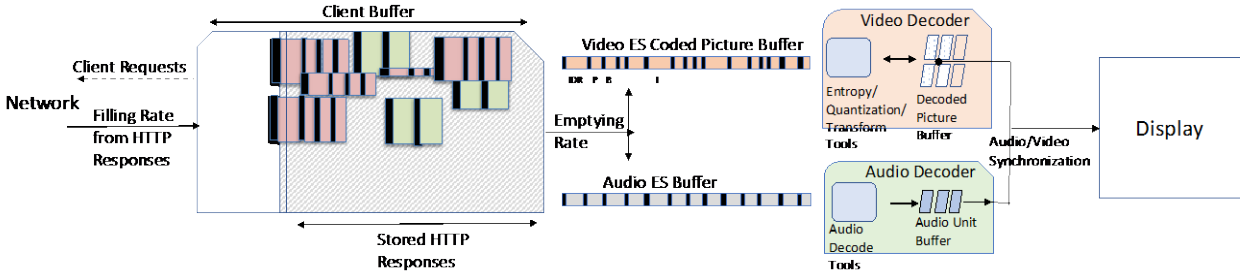

Figure 2: Player/decoder model.

The client buffer feeds the elementary stream's coded picture buffer (CPB) for both audio and video with the respective media segments. Depending on the algorithm, the loading of the CPB may be "chunky," and if not smooth enough, it may affect the HRD model's playout of the stream, which may also cause a rebuffering state. For video, the HRD model for a particular GoP (Group of Pictures) structure is maintained, from the ordering of the coded pictures, which improves video quality by retaining these pictures as a reference picture in the decoded picture buffer. A decoding latency occurs throughout this decoding process to typically accommodate a 2-4+ picture delay. Typically, gaining higher quality at the intended bitrate employs techniques at the cost of increasing latency. In fact, increasing quality of the stream or increasing scalability of the number of players happens with a tradeoff in latency (see Figure 3).

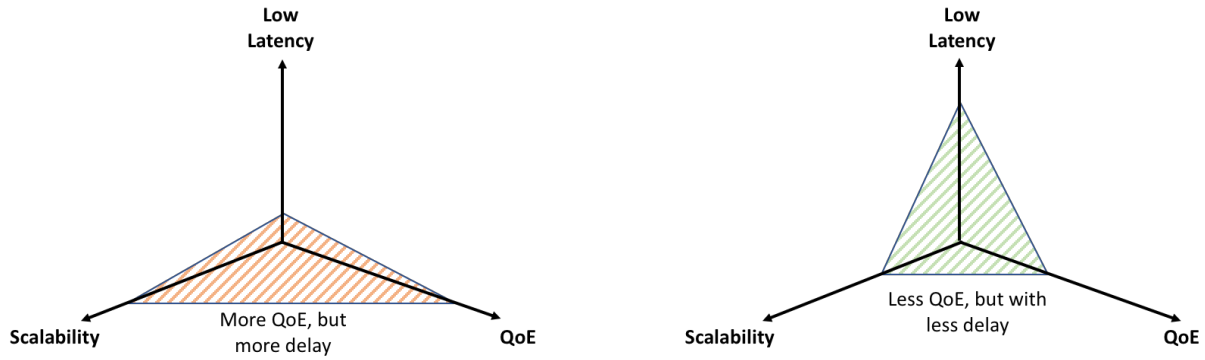

Figure 3 - Tradeoffs between low latency, scalability, and QoS (Quality of Service).

Additionally, shortening the segment size can reduce latency by filling in the CPB buffer more "smoothly" and adjusting the segment requests coming from the client. For example, if DASH (Dynamic Adaptive Streaming over HTTP) is used with 2-second segments, this results in latency of about 8-10 seconds (cumulative), while HLS (HTTP Live Streaming) using 10-second segments can result in latency as high as 30 seconds (assuming at least three segments are loaded into the buffer). With 6-second segments, which were allowable since mid-2016 for HLS, the latency for HLS streams reduced to 18-20 seconds. Thus, smaller segment durations can assist in reducing latency -- until the segments become too small and increase network traffic as a result of the increased requests to the server.

3. Types of Delivery

In the current fragment delivery models, the services all use different types of segmented, file-based delivery processes. Initially, on-demand applications were popular for ABR (Adaptive Bitrate)-enabled video services, because the creation of the content segment could be done well before the delivery. Furthermore, it could be prepositioned on the servers in the CDN, which shortened the delivery pathway from the server to the players.

Live events are another type of segmented file-based delivery process that differ from pre-packaged delivery because content segment creation is done serially with the delivery of those segments. In the live case, the viewing experience is sensitive to the latency from creation to delivery, while in the pre-packaged delivery case, it is more crucial to make sure the viewing quality is good relative to the latency considerations.

3.1. Pre-Packaged Delivery

For pre-packaged delivery, the encoded content already exists as a single file or set of segments. The goal for file-based delivery is for the client/player buffer to be filled such that it always allows the next frame to be decoded and played out. Delay in delivery of the next segment would cause the buffer to underflow, thus pausing the playout of the content -- to the obvious detriment of the QoE (Quality of Experience). Factors that can cause disruption in delivery include insufficient bandwidth between the server and client; congested bandwidth that causes delay of segments; unequal segment arrival times due to jitter; or simply missed segments traveling over a best-effort system that never arrive due to reaching the TTL (Time-to-Live) limit. Possible solutions to these issues include the following strategies:

- Create a longer startup time to allow for the CPB to grow larger before the playout process starts draining the buffer. This allows for more time to allow the next segment to reach the buffer (or to request an older, missing segment).
- Constrain the size of the segments being sent to avoid the risk of rapidly depleting the buffer. This can be done on a segment-by-segment basis, using adaptive streaming technologies.
- Monitor and manage bandwidth to the client by reducing the size of the segments. This can also be done by adaptive streaming technologies, where the player can monitor the buffer and request alternative and smaller-sized addressable content segments, of lower quality, in order to deal with best effort connections that at times experience bandwidth congestion. Other ways to avoid bandwidth congestion on the connection involve managing the capacity of the connection to mitigate bandwidth fluctuations.
- Create segments that are self-decodable, so as to avoid a loss because of segment corruption and the consequent need to retransmit it.

Pre-position segments at multiple servers situated physically closer to the client players, which can be done through a CDN network that can reduce the path latency of the packets. Having multiple servers can also reduce delays by reducing the number of segment requests at a single server during flash demand periods.

Each of these strategies can help to ensure smooth robust playback of the streams, but many of these plans create the need for a lengthened buffer that is more smoothly filled relative to both the client buffer and CPB. This adds to delay of the initial start of stream playback, and increases latency on the stream on the order of seconds or more.

3.2. Live Event Delivery

For live event delivery, the segments are created in the current moment by the camera capture and encoding processes. The segments, once created, are then transmitted over the network and delivered to the client player. It is meaningful, in live services, to reduce the glass-to-glass latency to reduce the amount of latency that may be more noticeable if alternative viewing distribution formats are available (e.g., MPEG2-TS (Moving Pictures Experts Group, Transport Stream) QAM (Quadrature Amplitude Modulation) delivery or attending the live event). There are several methods to reduce latency, which can even be cumulative, but each approach has tradeoffs in either latency, quality or robustness of playout. Methods to create lower latencies can be grouped in the following two categories:

- Modifying the segment creation process and providing better integration of this to the delivery process,
- Modifying the segment delivery process to bring this closer to the live edge. A lot of these techniques can be applied to file-based streaming as well.

In both of these categories, there are requirements wherein live streaming should still enable a recordable service (e.g., cDVR [Cloud Digital Video Recorder]) that can be used for later viewing of the content. Also, it is important to know that not all pre-packaged streaming features make sense for real-time streaming events.

4. Handling Low-Latency Live Events

The current adaptive streaming technologies have been able to handle live events, but at the cost of adding significant end-to-end delay. For next-generation technologies, a focus on reducing transmission latency is paramount. Such a latency reduction is well-served to focus on two areas: i) the content creation process for producing segments, and ii) the delivery of these segments to one or more players. In turn, some of these techniques, described below, can aid in pre-packaged segment delivery of content, too.

4.1. Segment Creation

To reduce latency in the area of segment creation, temporal compression techniques in the encoding process should be reduced. This will result in an alignment of the coded elementary stream with the presentation order of the frames, which can be done through the use of known spatial compression techniques, or very simplified FPP (Forward Predicted Picture) encoding processes. Both result in a loss of quality, so the existing and traditional techniques to improve quality (e.g., look ahead or pre-processing methods) need to be used sparingly to avoid adding latency to the stream. The resulting reduction in compression efficiency either results in a loss of quality or is compensated for by an increase in bitrate, which would increase the average connection bandwidth.

In terms of delivery of the in-process created segment, the packager does not have to wait for the entire segment to be processed, because it can release chunks (which do not need to be all self-decodable and could resolve to even a single frame) of the segment, which are then subdivided into decodable fragments [see Figure 4] as soon as they processed. This is a tool employed by the CMAF (Common Media Application Format) specification for ISO-BMFF media segments. This way, the application client can release the received chunks earlier to smoothly fill the player buffer as the chunks are being received. These chunks can also be playable as they are received, so long as the prior chunks for that segment have arrived. This approach can aid in filling the client buffer quickly, without waiting for the entire segment to be available for delivery.

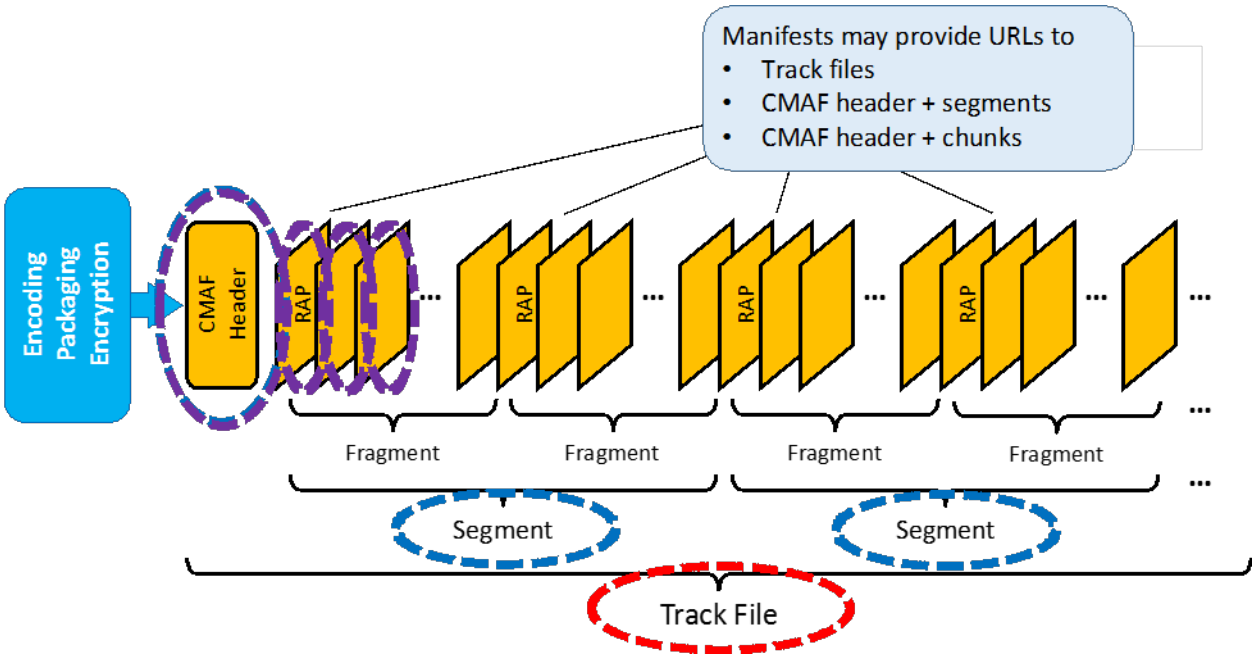

Figure 4: CMAF segment, fragment and chunk structures.

4.2. Segment-Based Delivery

Other opportunities to reduce latency are more applicable to how the segments are being delivered. In pre-packaged file delivery, prepositioning the segments beforehand, in the CDN system, may reduce delivery latency once the segments are requested. For live streaming this might not be as efficient. CDNs need to be cognizant of when they are handling a low-latency stream. Pre-positioning segments into more edge servers can cause start-up delays of a newly requested live and low-latency stream and can cause multiple players to not be synchronized in their playout. This can be attributable to delays with manifest creation/segment availability, or even just the delay of copying and distributing segments on multiple cascaded servers in the delivery path. Moving more towards a centralized server distribution or minimal hierarchy layered distribution can reduce this type of delay but would extend path latency issues.

An alternative to using a full CDN distribution mechanism to reduce live steam latency is to stream from a central server, but in this case, scalability would become an issue. If a low-latency stream is highly requested, then the central server/server(s) would need to be scaled to respond to a high traffic volume of request/responses from thousands or even hundreds of thousands of clients. An alternative approach would be to use a type of multicast/broadcast transmission to a player or an edge server. Using an approach that multicasts segment transmission, such as ATSC (Advanced Television Systems Committee) 3.0, could handle the traffic volume, but potentially at a latency cost, where the latency sacrifices come from delays in manifest creation and segment availability. Another alternative approach could be a hybrid of this, where content is multicast to local servers and then distributed from that point. The multicast component does not have to be a multicast of segments but could instead be a multicast of a marked up MPEG2-TS stream using AF, or Adaptation Field Descriptors, as described in the MPEG-2 Systems document and using the manifest described in SCTE 214-4, which is sent to a local server or even at a home gateway and packaged at that point. These types of streams can be automatically outputted from the encoder and traditionally multicast to a local server without the need for a manifest. A linear packager at the local server can then create the segments and the manifest locally. As one can see, there are several different approaches that could reduce the distribution latency to the servers.

Another area to look for possible latency reduction is the internet transport protocols for handling information exchange. At the application layer, HTTP/1.1 is commonly used for server- to-client exchanges of media. The HTTP/1.1 protocol may use a free and open-source web server implementation. Using the HTTP protocol allows for adaptive streaming across generalized IP (Internet Protocol) hardware (using port 80) without designing specifically for each piece of hardware; this is widely known as HTTP adaptive streaming (HAS). Modifying HTTP can solve for specific problems of adaptive streaming in HTTP, but the cost is deviating from an open-source implementation and needing both the transmission and receiving points to download additional code to handle this modified connection -- which is hard to implement consistently in an open general IP environment.

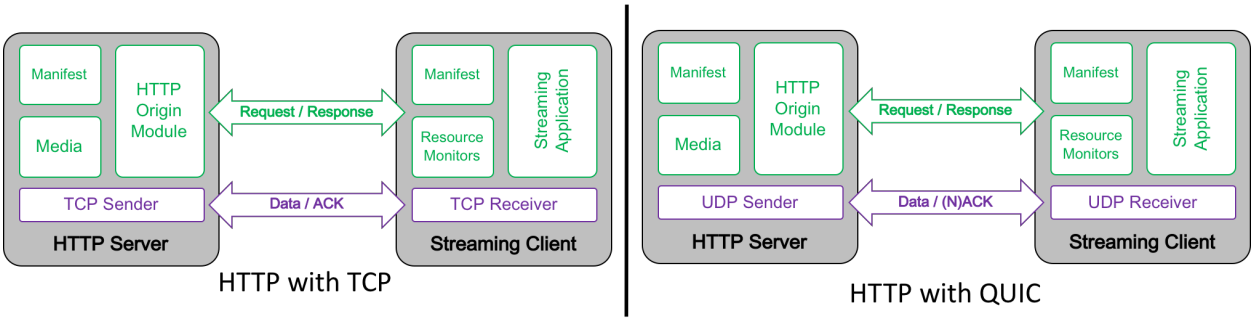

Figure 5: Server distribution using HTTP over TCP and QUIC.

A lower layer area that could be focused on would be the transport layer protocol. The protocol typically used with HTTP is TCP (Transmission Control Protocol) (see Figure 5), which is a protocol between two nodes to set up a connection and then send data packets over the internet. This type of protocol ensures that all data packets are received, in order, with a method to detect missing packets and request that they be sent again. For low-latency streaming, two features of TCP are highly sensitive to low-latency streaming. The first factor is that the re-request of a data packet must come in time to be useable by the buffer. With a low buffer for low latency, there are greater chances that the packet may not arrive in time, which means that the playability of the stream would get interrupted -- so this operation would fail more in low-latency situations anyways.

The second factor is that the packets must be received, in order, in the application buffer, before they are released to the player buffer (head-of-line blocking). These factors increase as best effort systems encounter more bandwidth congestion. There are several approaches to deal with this, but require a change in the protocol. An immediate approach would be to open up several TCP connections for each exchange, such that head-of-line blocking could be avoided. Another approach is to use UDP (User Datagram Protocol) instead of the TCP protocol, where UDP benefits because it does not attempt to even request a missing data packet.

An alternate but increasingly popular protocol that takes advantage of this is QUIC (Quick UDP Internet Connections), where the initial round-trip exchanges to set up are reduced (see Figure 5). Other approaches would be to send multiple data packets upon a single response with only negative acknowledgements sent. Thesetypes of modifications are being developed in new specifications such as the QUIC protocol and in the future HTTP/2 versions. Both versions require moving away from an already accepted open-source web server approach, and for that reason, acceptance of the improved protocol, and transitioning to them, may take time. Another way to mitigate these types of factors is to avoid bandwidth congestion or reconnections where possible. This can be done by adapting the bandwidth environment with respect to the particular service of the stream, which can then be handled in managed bandwidth environments more easily than in best-effort systems.

For low-latency live streaming, other alterations that would help would be to move closer to the live edge of delivery. For pre-packaged delivery, the priority is to provide robust streams to players to create an uninterrupted viewing experience, which involves requesting segments that are three segments behind the live edge. This ensures there is always a segment sent for each request, which reduces the risk of rebuffering. In live low-latency streaming, the priority may be to receive packets as quickly as possible, and the uninterrupted experience may be secondary to that. Using this premise, the use of manifests may change. For instance, the three-segment delay (to ensure availability of the segment) for manifests in adaptive streaming could be shortened. The use of predictive templates may be useable where the client may request future segments that have not been published yet, but will be. This, combined with error handling techniques at the player, may provide a useable live low-latency stream. Some of these player techniques to avoid rebuffering situations could be to use to repeat frames, skip frames, or apply temporary slowing of the frame rate, where the low latency of the stream is treated as a priority.

Conclusion

Reducing latency will improve the viewing experience in live event streaming. In current generation adaptive streaming situations, which were more designed to enable pre-packaged on-demand streaming, the priority was to ensure the robustness of the played stream -- which may add latency to the system, but improves the overall QoE. With a renewed focus on live streams, a balance needs to exist between robustness of the stream and a reduction of end-to-end latency.

There are three areas to reduce latency: 1) segment creation, 2) distribution, and 3) playout; it's a tradeoff between reducing latency, minding delivery quality, and accommodating scalability (especially for live events, which often generate flash crowds). Reducing latency cannot be done with just one change: It is a cumulative sum of incremental changes in all these areas. For content creation, it's about how to encode/decode the ES stream faster and yet still maintain quality. That requires integrating segment creation into delivery and distributing it quickly into the network, through a series of chunks that can be re-assembled into a segment or fragment. In distribution, it is a matter of how to get it to the edge servers quickly, while shortening the request/response communication between server/client, through the use of multiple TCP connections, or by moving to UDP-types of connections. Lastly, on the player side, the player should recognize that this is a low-latency stream where it is okay to anticipate segment requests, adjust the buffer for shorter durations, and allow for new playout behavior to keep the stream playable in a low-latency mode.

Working on reducing latency in the system will help in live event streaming, but will also generally aid in all segment-based deliveries (including pre-packaged delivery), by moving distribution closer to the live edge.

It is to the benefit of next-generation IPTV systems to integrate low-latency streaming into part of their design. This will improve performance of the next-generation IPTV systems.

Abbreviations

ABR	Adaptive Bitrate
AF	Adaptation Field

ATSC	Advanced Television Systems Committee
AVC	Advanced Video Coding
CDN	Content Delivery Network
cDVR	Cloud Digital Video Recorder
CMAF	Common Media Application Format
CPB	Coded Picture Buffer
DASH	Dynamic Adaptive Streaming over HTTP
DPB	Decoded Picture Buffer
ES	Elementary Stream
FFWD	Fast Forward
FPP	Forward Predicted Picture
GoP	Group of Pictures
HAS	HTTP Adaptive Streaming
HEVC	High Efficiency Video Coding
HLS	HTTP Live Streaming
HRD	Hypothetical Reference Decoder
I/O	Input/Output
IP	Internet Protocol
IPTV	Internet Protocol Television
ISBE	International Society of Broadband Experts
ISOBMFF	ISO Base Media File Format
MPEG	Moving Pictures Experts Group
QAM	Quadrature Amplitude Modulation
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RWD	Rewind
TCP	Transmission Control Protocol
TS	Transport Stream
TTL	Time To Live
UDP	User Datagram Protocol
VoD	Video on Demand
SCTE	Society of Cable Telecommunications Engineers
NCTA	National Cable & Telecommunications Association

Bibliography & References

ISO/IEC 23009-1 Information Technology- Dynamic adaptive Streaming over HTTP (DASH) Part 1: Media presentation description and segment formats – Second Edition

ISO/IEC 23000-19, Information Technology- Multimedia application format (MPEG-A)- Part 19: Common media application format (CMAF) for segmented media- First Edition

ISO/IEC 13818-1:2018, Information Technology – Generic coding of moving pictures and associated audio – Part 1: Systems

SCTE 214-4 2018, MPEG DASH for IP-Based Cable Services Part 4: SCTE Common Intermediate Format (CIF/TS) Manifest for ATS Streams

SCTE 223 2017, Adaptive Transport Stream

Category-aware Hierarchical Caching for Video-on-Demand Content on YouTube. Christian Koch, Johannes Pfannmuller, Amr Rizk, David Hausheer, Ralf Steinmetz, MMSYS'18, June 12-15, 2018, Amsterdam, Netherlands

E. Thomas, R. Koenen, A. Begen, J. Boyce, "Streaming-First design for the MPEG-I project", M43753-MPEG Meeting Documents, July 2018, Ljubljana, SI

A.Langley, A. Riddoch, A.Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti,, R. Shade, R. Hamilton, V. Vasilev, W. Chang, Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment", SIGCOMM '17 August 21-25, 2017 Los Angeles, CA, USA

A. Begen, C. Timmerer, L. Ma, "Delivering Traditional and Omnidirectional Media", ICME Tutorial, ICME 2018, San Diego, CA

T. Stockhammer, "Low-Latency DASH", MHV 2018, Denver, CO, July 31st-Aug. 1st, 2018

Y. Shen, "Low-Latency Live Streaming at Scale", MHV 2018, Denver, CO, July31st-Aug. 1st, 2018

