



# Advances in Multimedia Streaming: Algorithms, Standards and Optimization Techniques

*IEEE ICIP Tutorial – Sept. 19<sup>th</sup>, 2021*



**Ali C. Begen, PhD and Yuriy A. Reznik, PhD**

ali.begen@ozyegin.edu.tr, yreznik@brightcove.com

# Upon Attending This Tutorial, You will Know about

- History of streaming, key problems and innovations
- HTTP adaptive streaming: DASH, CMAF and HLS, and deployment workflows
- Areas of recent progress
- Ongoing research and open issues

This tutorial is public, however, some material might be copyrighted  
Use proper citation whenever you use content from this tutorial

*Many thanks to my colleagues; to name a few: W. Law, C. Timmerer, T. Stockhammer, Z. Cava, Y. Syed, C. Concolato, A. Giladi, R. Mekuria, J. Piesing, D. Silhavy, J. Simmons, I. Sodagar*

# Presenters Today

## Ali C. Begen (<https://ali.begen.net>)

- Focus and research areas
  - Protocols and algorithms for multimedia applications
  - Modeling/prototyping and analysis of networked systems
  - Network programming and support for multimedia
- Education and experience
  - BSEE @Bilkent U. (2001), PhD @Georgia Tech (2006)
  - Associate prof. @OzU and IEEE Distinguished Lecturer
  - Nine years of research, development at Cisco
  - Five years of consulting for Comcast/NBCU, InterDigital, MediaMelon, Turkcell, Digiturk and legal firms
  - 30+ granted US and international patents



## Yuriy A. Reznik (<http://reznik.org>)

- Focus and research areas
  - Speech, Audio, and Video processing, perceptual quality metrics, perceptual coding
  - Large scale multimedia streaming systems
  - Cloud-based systems and workflows
- Education and experience
  - PhD, CS @ Kiev University (2005)
  - Visiting Scholar at Stanford University (2008)
  - Co-developer of first ABR system (RealSystem G2 – 1998)
  - Contributor to several MPEG and ITU-T standards: H.264/AVC, MPEG-4 ALS, G.718, H.265/HEVC, DASH, etc.
  - 20+ years in industry (RealNetworks, Qualcomm, InterDigital, Brightcove)
  - 70+ granted US patents





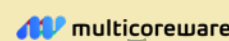
Sept. 28-Oct. 1, 2021 – Istanbul, Turkey

Bridging Deep Media and Communications

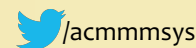
**MMSys, NOSSDAV, MMVE, GameSys and Much More**

<http://2021.acmmmsys.org>

*Supported by*



\* Including the postponed celebrations for the 30<sup>th</sup> and 25<sup>th</sup> anniversary of NOSSDAV and PV, respectively





# Call for Contributions – ACM Mile High Video 2022

## *A new ACM SIGMM conference*

- Unique forum for participants from both industry and academia
- Innovations from content production to consumption
  - Content production, encoding and packaging
  - Workflows
  - Content delivery and security
  - Streaming technologies
  - Industry trends
  - Standards and interoperability
- Extended abstracts due Oct. 1
- Details at <https://mile-high.video>



# Outline

- History of streaming, key problems and innovations
- HTTP adaptive streaming: DASH, CMAF and HLS, and deployment workflows
- Areas of recent progress
  - Improvements in client playback and rate-adaptation algorithms
  - Low-latency live streaming extensions
  - Content preparation for streaming: general considerations
  - Per-title, content and context-aware encoding
    - Optimizations for multi-codec streaming
    - Optimizations for multi-screen streaming
    - HTTP streaming and CDN performance
- Ongoing research and open issues
  - AI/ML optimizations for streaming
  - Quality-aware streaming
- Q&A

# History of Streaming, Key Problems and Innovations



# A Look at a Longer Timeline

## Evolution of video technologies

### THE PAST:

Invention of camera, still image photography, color reproduction, film, moving pictures

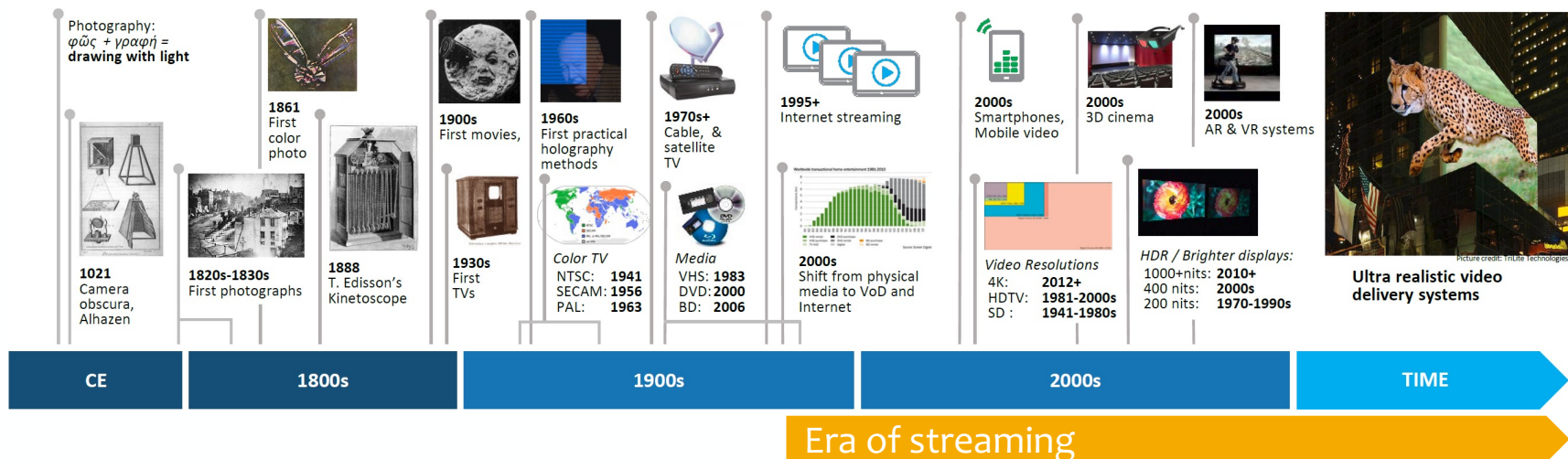
### THE PRESENT:

New delivery methods: TV, recordable media, digital compressed formats, Internet streaming, mobile.

Increasing degree of realism: immersive video, 3D (holography, stereoscopic rendering, etc.)

### THE FUTURE:

Recording & reproduction systems making rendered video undistinguishable from reality.



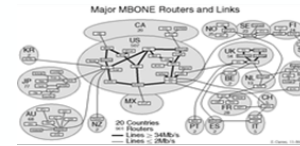
# Early Streaming Systems

## 1993: MBONE

- Virtual multicast network connecting several universities & ISPs
- RTP-based video conferencing tool (vic) is used to stream videos
- 1994 Rolling Stones concert – first major event streamed online

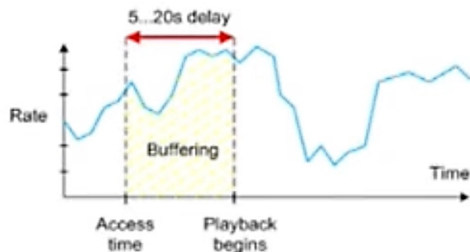
## 1995: RealAudio, 1997: RealVideo

- First commercially successful mass-scale streaming system
  - Proprietary protocols, codecs: PNA, RealAudio, RealVideo
  - Worked over UDP, TCP, and HTTP (“cloaking” mode)
  - First major broadcast: 1995 Seattle Mariners vs New York Yankees
- 1996+: VDOnet, Vivo, NetShow, VXtreme, ...
    - Many vendors have competed in streaming space initially
    - Vivo & Xing have been acquired by Real, VXtreme by Microsoft
    - By 1998 only Real and Microsoft remained, joined by Apple in 1999
  - 1998: RealSystem G2
    - First ABR streaming system



# First Important Innovations in Streaming

- Discovery of a pre-roll delay
  - Many early systems (Vivo, VDOnet, etc.) have tried to use H.324 / H.323- video conferencing stacks for streaming. But they worked very poorly!
  - The first important discovery and deviation in the design of streaming systems from video conferencing was introduction of a much longer initial delay.
- Original uses of the pre-roll delay / buffer
  - Leaky bucket model: reducing probability of stalls with network bandwidth fluctuations
  - Reordering of out-of-order received UDP packets
  - Limited retransmissions (ARQ)
  - Interleaving / multiple-description coding of audio
- Interleaved packetization (RealAudio, 1995)
  - 20-ms audio frames after encoder
  - Missing audio frames were by-directionally predicted/synthesized during decoding
  - This worked remarkably well even with heavy (5-10%) packet loss rates!

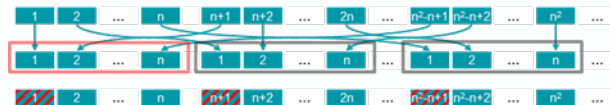


Expected delay, throughput and throughput distribution caused by retransmissions:

$$\bar{T}(A, B) = \sum_{i=0}^{\infty} (1-p)p^i (1+i)\tau = \frac{\tau}{1-p}$$

$$\bar{R}(A, B) = \sum_{i=0}^{\infty} (1-p)p^i \frac{N}{(1+i)\tau} = \frac{N(1-p)}{\tau p} \log\left(\frac{1}{1-p}\right)$$

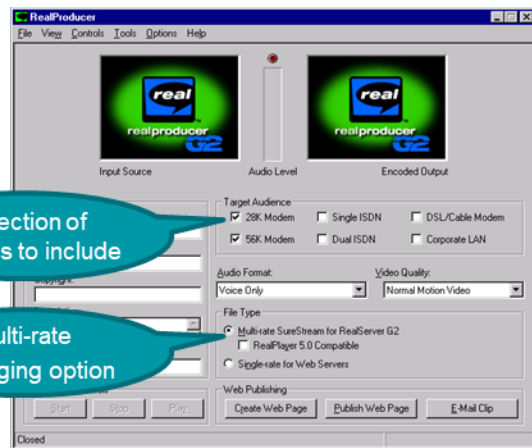
$$\Pr\left(R = \frac{N}{(1+i)\tau}\right) = (1-p)p^i$$



# First ABR Streaming System

- 1998: RealSystem G2, “SureStream” technology
  - First commercially successful ABR streaming system

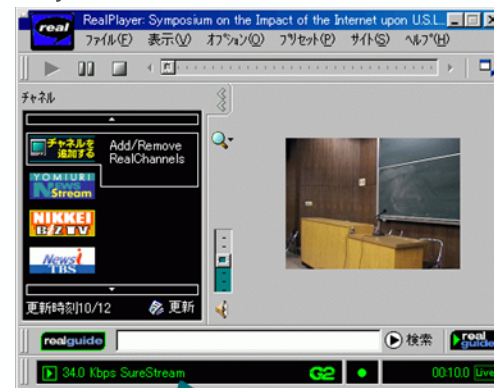
Encoder



Encoded streams



Player



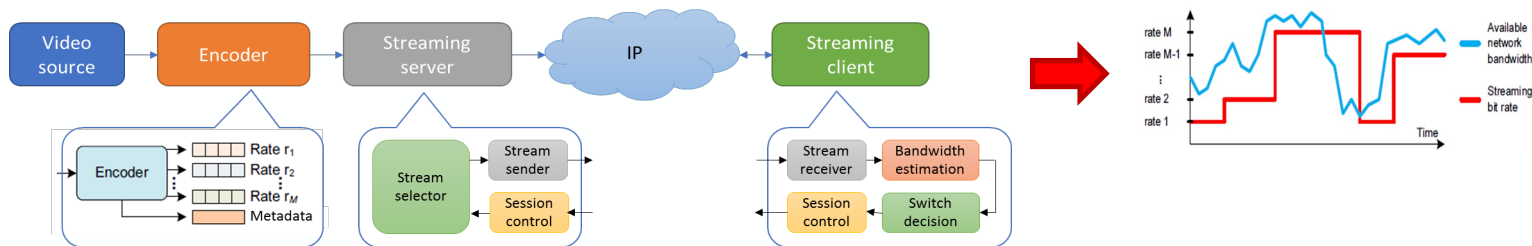
Panel showing which stream is selected

- Related publications and patents
  - A. Lippman, “Video coding for multiple target audiences,” VCIP’98
  - G. Conklin, et al. “Video Coding for Streaming Media Delivery on the Internet,” TCSVT, 2001
  - US patents: 6314466, 6480541, 7075986, 7885340



# How First ABR Streaming System Worked?

- RealSystem G2 architecture (1998)
  - RTSP session control, RDT, RTP, or TCP for stream transmissions (Public IP is used for delivery)
  - Stream adaptation was done by server, but it was client-driven: client was sending requests to switch
  - Server was also responsible for retransmissions, mixing in FEC packets, etc.
  - Everything was sent in “packets”

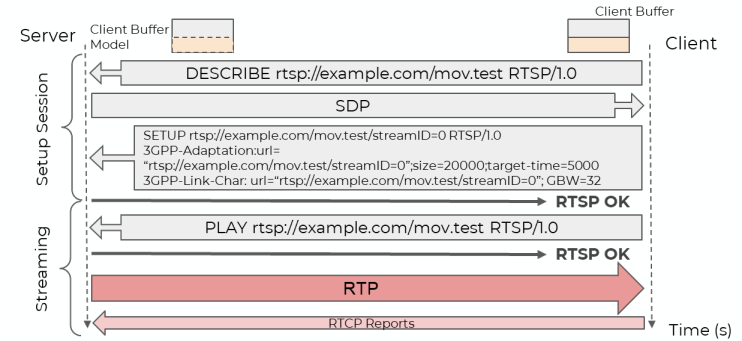


- Challenges faced
  - Scalability: Each single server was scalable to ~10K streams, and to scale beyond you needed another server
  - Reliability: Public IP was horrible, as were last mile links; reliability of the servers was also limited
  - Implementation complexity: Advanced error resilience and error concealment functionality was required for every decoder in every client. All codecs had to be custom written and supported on all platforms.

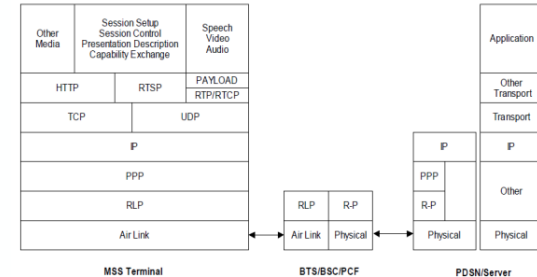


# Early Streaming Standards

- **1998: RTSP – Real-Time Streaming Protocol**
  - Session protocol for packet-bases streaming
  - Main contributors: RealNetworks, Netscape, Columbia University
  - Uses as foundation for most streaming systems of 1998-2008 era
- **2000: ISMA – Internet Streaming Media Alliance**
  - Forum created by Apple, Cisco, Kasenna, Philips, and Sun
  - ISMA 2.0: RTSP+RTP+RTCP + H.264 and HE-AAC codecs
  - ISBMFF with hint tracks is employed for storage of encoded streams
  - ISMA 2.0 was supported by many servers and clients of that era
- **2006: 3GPP PSS – Packet Switched Streaming**
  - Describes RTSP+RTP+RTCP ABR adaptive streaming system with several standard video, audio and speech codecs
  - 3GPP version of RTSP/RTP-based stack
- **2006: 3GPP2 MSS – Multimedia Streaming Services**
  - Similar to 3GPP PSS, but differs in speech codecs & network stack



Full protocol stack in 3GPP2 MSS



# Why Today's Streaming Systems Use HTTP?

- Networks have improved!!
  - When streaming started, 28k and 56k modems were the common connections available
  - But by mid-2000s consumers moved to Cable, DSL, or other high-speed connections
  - Bitrates were up 5-100x, latencies were 4-10x down, packet losses were under 1-2%
  - This relaxed requirements dramatically!
  - Progressive downloads become feasible alternatives to streaming!
- CDNs became ubiquitous
  - By mid-2000s Akamai, Limelight and other CDNs were well deployed all over
  - CDNs provided much better density and reach than RTSP-based delivery networks (RBN, etc.)
- Other practical and business reasons
  - The space was fragmented: Real, Microsoft, Apple, and then Adobe used significantly different implementations of their stacks. Even codecs and file formats were different! RTSP and ISMA offered only “baseline” level of interoperability!
  - RTSP systems were complex: servers and clients were extremely complex, error concealment was a major pain, etc.
- And ... one day a much simpler solution was found
  - Store encoded media streams in 5-10sec chunks on a web server... pull them using HTTP GET, concatenate and play
  - About same delays, no packet losses or retransmissions, and with good enough networks – it may just work...



# Further Reading on Early Designs of Streaming Systems

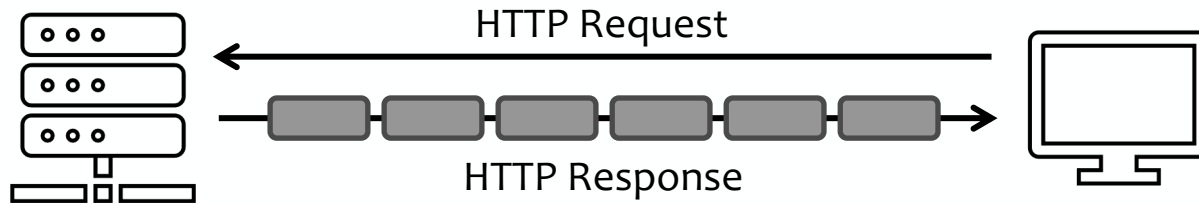
- B. Girod, N. Farber, and U. Horn, “Scalable Codec Architectures for Internet Video-on-Demand,” in Proc. Thirty-First Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 1997, pp. 357-361
- N. Farber and B. Girod, “Robust H.263 compatible video transmission for mobile access to video servers,” Proc. International Conference on Image Processing, Santa Barbara, CA, 1997, pp. 73-76
- A. Lippman, “Video coding for multiple target audiences,” Proc. SPIE 3653, VCIP'99, 28 Dec. 1998
- G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik, “Video Coding for Streaming Media Delivery on the Internet,” IEEE Trans. CSVT, 2001, 11 (3), pp. 20-34
- D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha, “Streaming Video Over the Internet: Approaches and Directions,” IEEE Trans. Circuits Syst. Video Technol., 11(3):282–300, Mar. 2001
- B. Girod, M. Kalman, Y.J. Liang, and R. Zhang, “Advances in channel-adaptive video streaming,” Wireless Comm. and Mobile Comp., vol. 2, no. 6, pp. 573–584, 2002.
- Y. Reznik and K. Lillevold, “History of stream switching: S-frames, SI/SP-frames, leaky prediction, gradual refresh, HESP, etc.,” MPEG input contribution M56085, January 2021

# HTTP Adaptive Streaming: DASH, CMAF and HLS, and Deployment Workflows



# Progressive Download

*One request, one response*



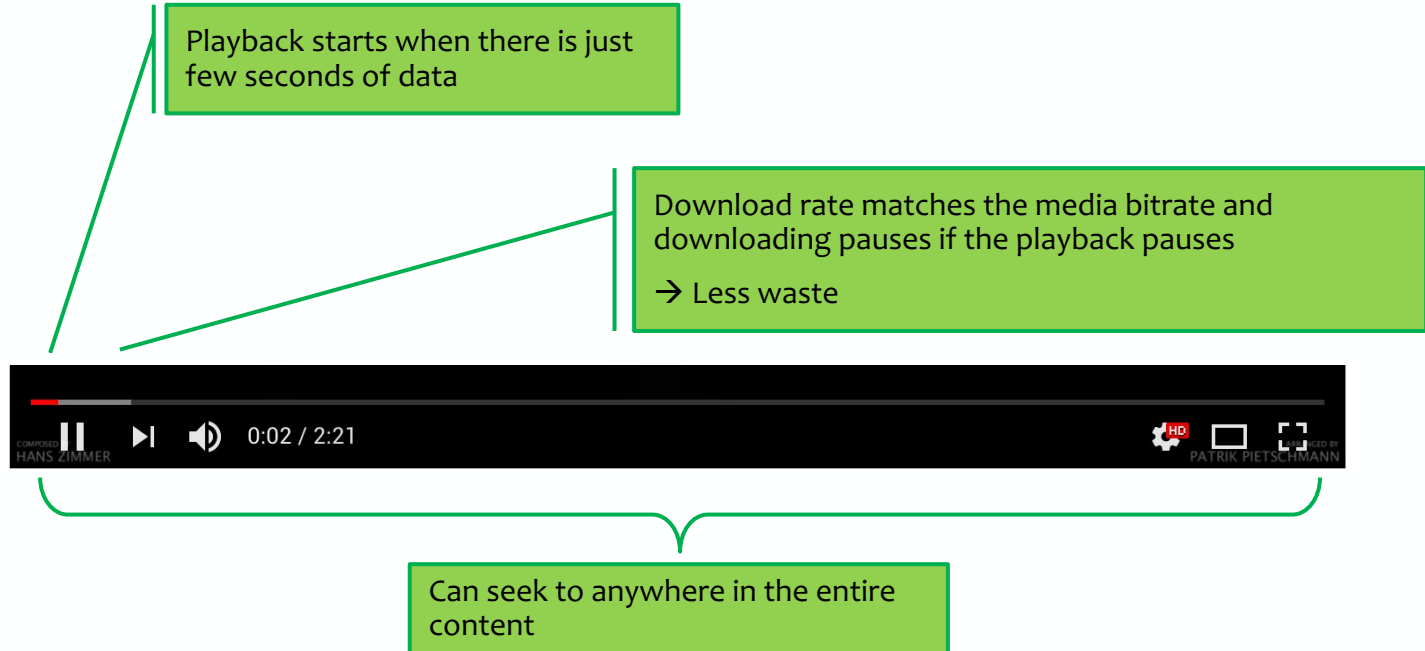
Playback starts only after there is several seconds of data in the playback buffer

Download will continue as fast as possible  
→ Unwatched content (waste)



Can seek only throughout the fetched content – not further

# Streaming is More Viewer and Network Friendly



## What is streaming?

- Transmission of a continuous media
- Simultaneous consumption by the client
- *No download-and-play*

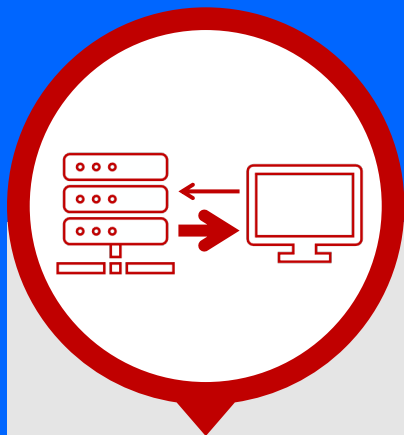
## Client consumption rate

- Limited by bandwidth availability
- Also limited by real-time constraints
- *Client cannot fetch content that is not available yet*

## Server transmission rate

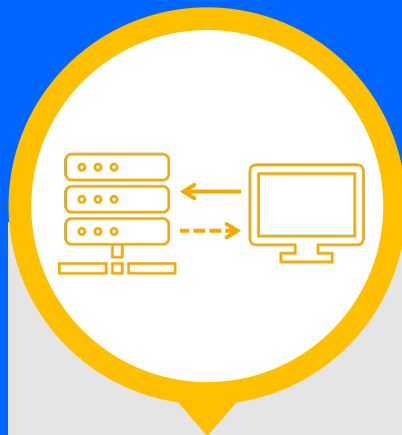
- Loosely matches to client consumption rate (long buffer)
- Tightly matches to client consumption rate (short buffer)
- *No buffer overrun underrun is acceptable*

# Delivering Media over HTTP beyond Download-and-Play



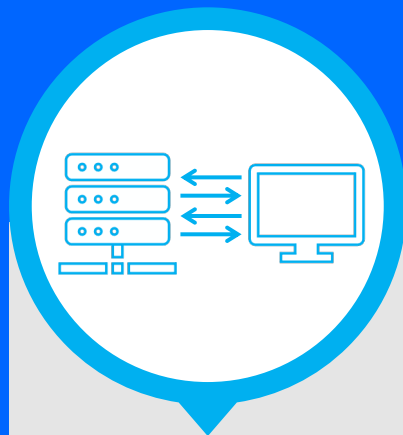
## Progressive Download

Playing while still downloading  
No throttling



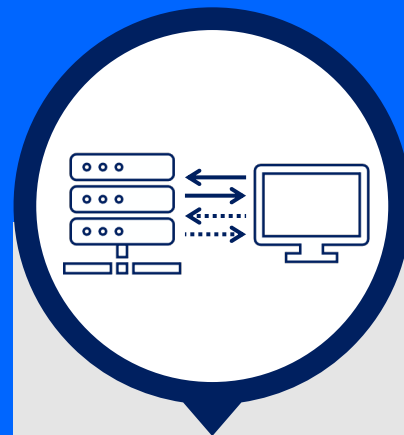
## Pseudo Streaming

+ Seeking  
+ Throttling based on encoding bitrate



## Chunked Streaming

+ Live and linear streaming  
+ Ad insertion

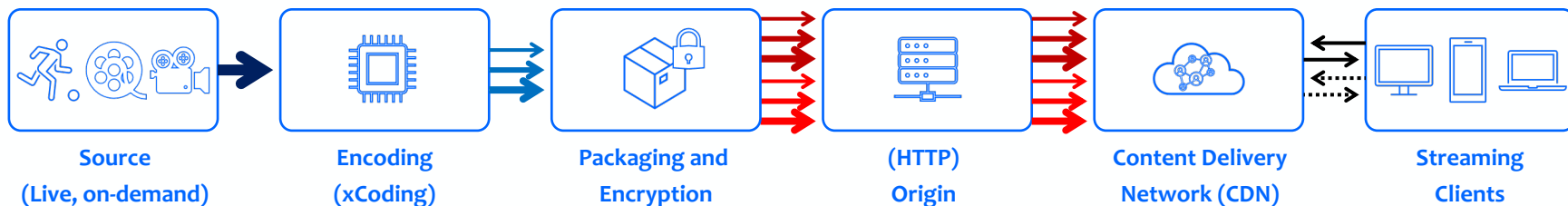


## Adaptive Streaming

+ Adapting to network and client status



# End-to-End Workflow for HTTP Adaptive Streaming



- Why HTTP
  - Features well-understood naming/addressing and authentication/authorization infrastructure
  - Provides easy traversal for all kinds of middleboxes (e.g., NATs, firewalls)
  - Enables cloud access, leverages the existing (cheap) HTTP caching infrastructure
- Imitation of streaming via short downloads (request/response pairs)
  - Minimizes (download) waste
  - Enables monitoring/tracking consumption
- Improved viewer experience
  - Reduces startup delay (upon zapping or seeking), frame skips and stalls
  - Provides adaptation capability based on network conditions and client status

# Dead, Surviving and Maturing Technologies, and Others

- Move Adaptive Stream (Long gone)
  - Some components are in Slingbox)
- Microsoft Smooth Streaming (Legacy)
  - <https://azure.microsoft.com/en-us/services/media-services>
- Adobe Flash (Dead)
  - <http://www.adobe.com/products/flashplayer.html>
- Adobe HTTP Dynamic Streaming (Legacy)
  - <http://www.adobe.com/products/httpdynamicstreaming>



- Apple HTTP Live Streaming (The elephant in the room)
  - <https://tools.ietf.org/html/rfc8216>
  - <https://datatracker.ietf.org/doc/draft-pantos-hls-rfc8216bis>
- DASH and CMAF (The standards)
  - <https://www.mpegstandards.org/standards/MPEG-DASH/>
  - <https://www.mpegstandards.org/standards/MPEG-A/19/>

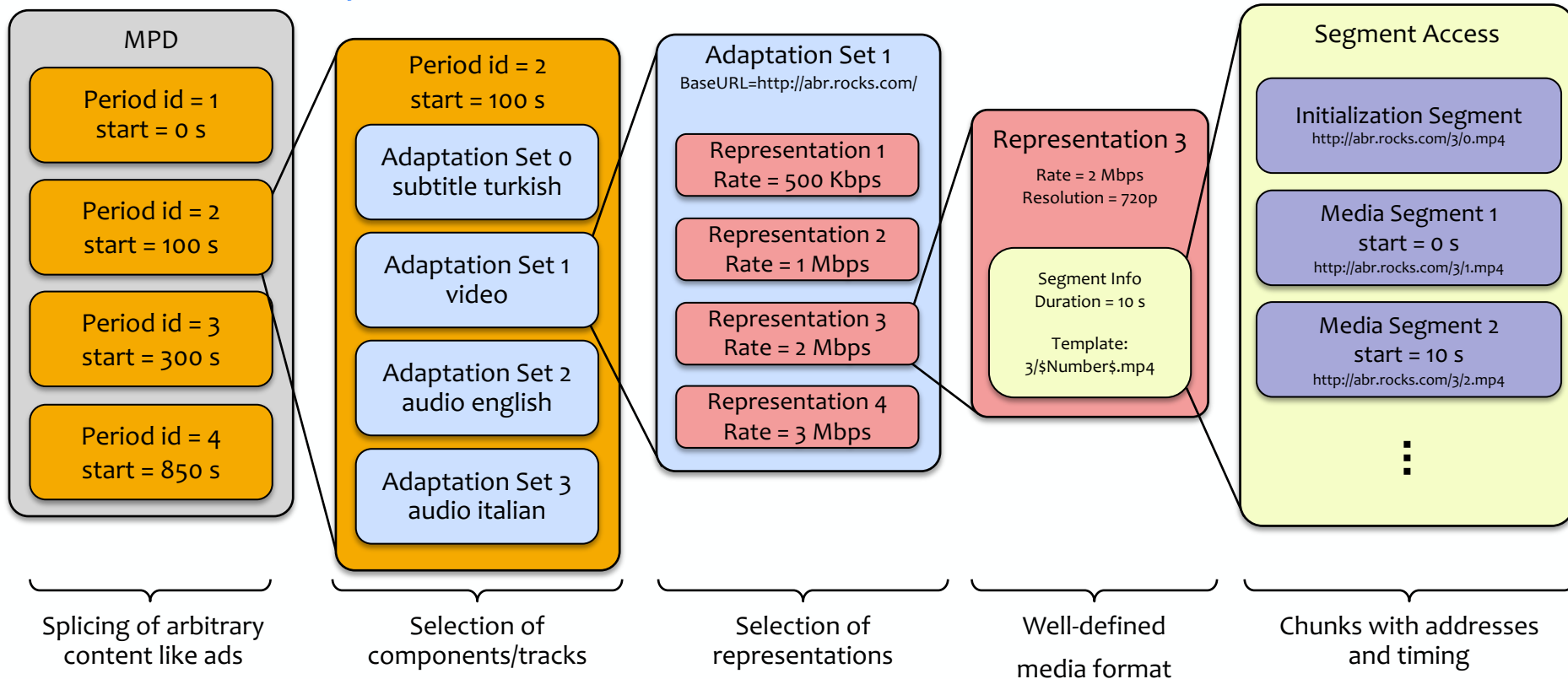


- YouTube and Netflix
  - Some standard, some proprietary components



# Manifests: List of Accessible Segments and Their Timings

*DASH MPD: Template based and extensible*



# Manifests: List of Accessible Segments and Their Timings

*HLS Playlist: Text based and verbose*

**master.m3u8**

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=232370,CODECS="mp4a.40.2"
gear1/prog_index.m3u8

#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=649879,CODECS="mp4a.40.2"
gear2/prog_index.m3u8

#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=41457,CODECS="mp4a.40.2"
gear0/prog_index.m3u8
```

**gear1/prog\_index.m3u8**

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:9.97667,
fileSequence0.ts
#EXTINF:9.97667,
fileSequence1.ts
#EXTINF:9.97667,
fileSequence2.ts
.
.
.
#EXT-X-ENDLIST
```

Source: <https://developer.apple.com/streaming/examples/> and <https://www.gpac-licensing.com/2014/12/01/apple-hls-comparing-versions>

# Example Representations

## Vancouver 2010

	Encoding Bitrate	Resolution
<b>Rep. #1</b>	3.45 Mbps	1280 x 720
<b>Rep. #2</b>	1.95 Mbps	848 x 480
<b>Rep. #3</b>	1.25 Mbps	640 x 360
<b>Rep. #4</b>	900 Kbps	512 x 288
<b>Rep. #5</b>	600 Kbps	400 x 224
<b>Rep. #6</b>	400 Kbps	312 x 176

## Sochi 2014

	Encoding Bitrate	Resolution
<b>Rep. #1</b>	3.45 Mbps	1280 x 720
<b>Rep. #2</b>	2.2 Mbps	960 x 540
<b>Rep. #3</b>	1.4 Mbps	960 x 540
<b>Rep. #4</b>	900 Kbps	512 x 288
<b>Rep. #5</b>	600 Kbps	512 x 288
<b>Rep. #6</b>	400 Kbps	340 x 192
<b>Rep. #7</b>	200 Kbps	340 x 192

## PyeongChang 2018

	Encoding Bitrate	Resolution
<b>Rep. #1</b>	18 Mbps	4K (60p)
<b>Rep. #2</b>	12.2 Mbps	2560x1440 (60p)
<b>Rep. #3</b>	4.7 Mbps	2K (60p)
<b>Rep. #4</b>	3.5 Mbps	1280x720 (60p)
<b>Rep. #5</b>	2 Mbps	1280 x 720
<b>Rep. #6</b>	1.2 Mbps	768 x 432
<b>Rep. #7</b>	750 Kbps	640 x 360
<b>Rep. #8</b>	500 Kbps	512 x 288
<b>Rep. #9</b>	300 Kbps	320 x 180
<b>Rep. #10</b>	200 Kbps	320 x 180

Source: Vertigo MIX10, Alex Zambelli's Streaming Media Blog, Akamai, Comcast

# HAS Working Principle

## *Smart and selfish clients*



- Client fetches and parses the manifest
- Client uses the OS-provided HTTP stack (HTTP/1.1/2 runs over TCP, HTTP/3 runs over QUIC)
- Client uses the required decryption tools for the protected content

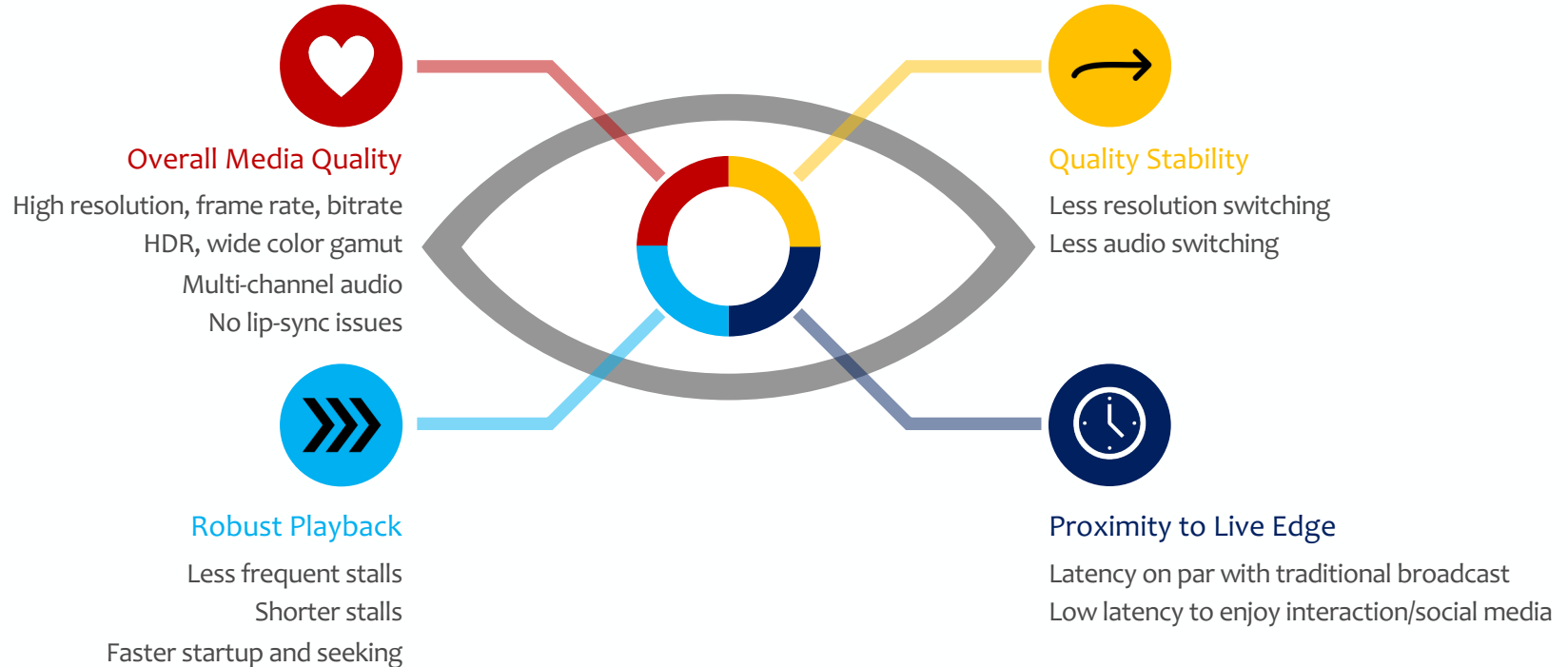
### Client monitors and measures

- Size of the playout buffer (in bytes and/or seconds)
- Chunk download times and bandwidth
- Local resources (CPU, memory, window size, etc.)
- Dropped frames

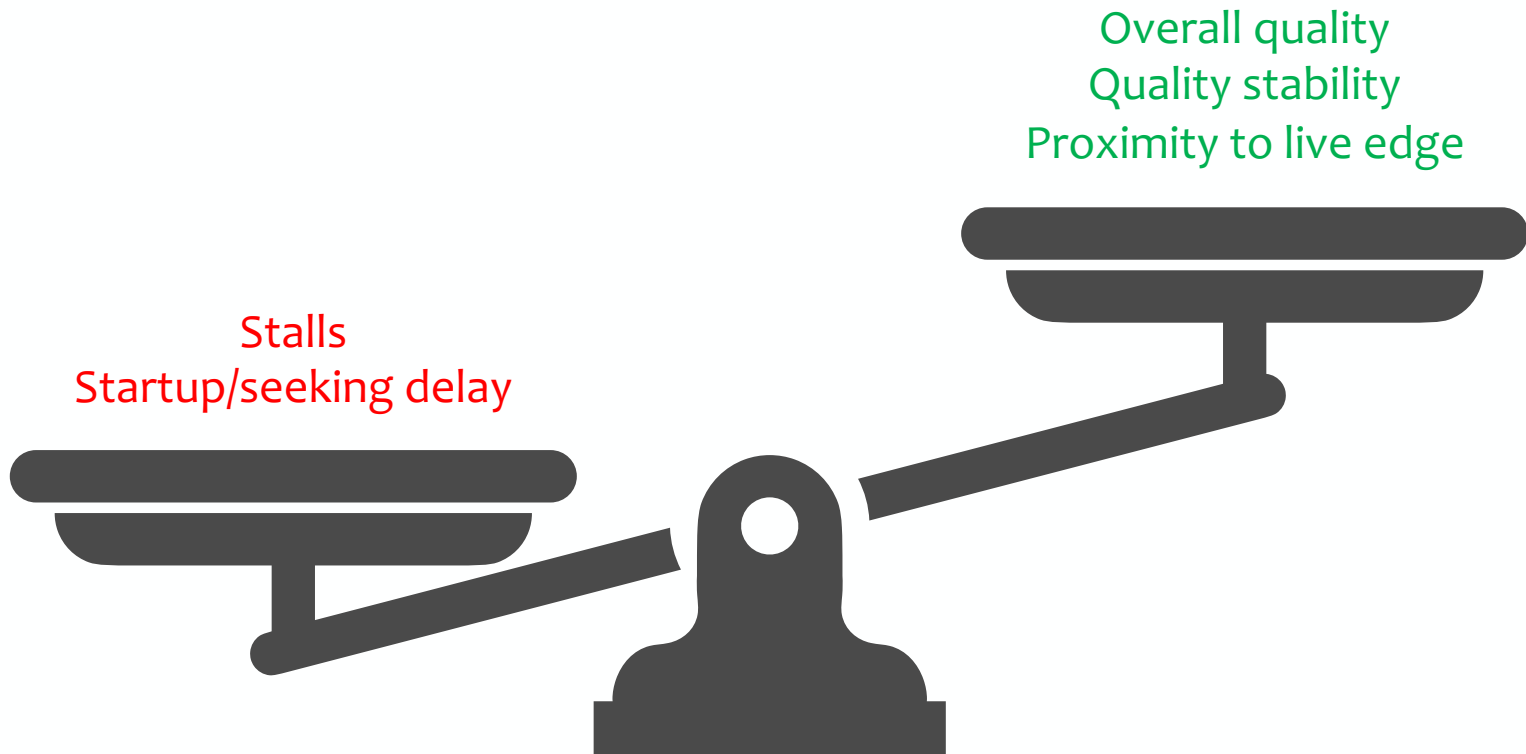
### Client performs adaptation

### Client measures and reports metrics for analytics

# What Matters to Human Eye?



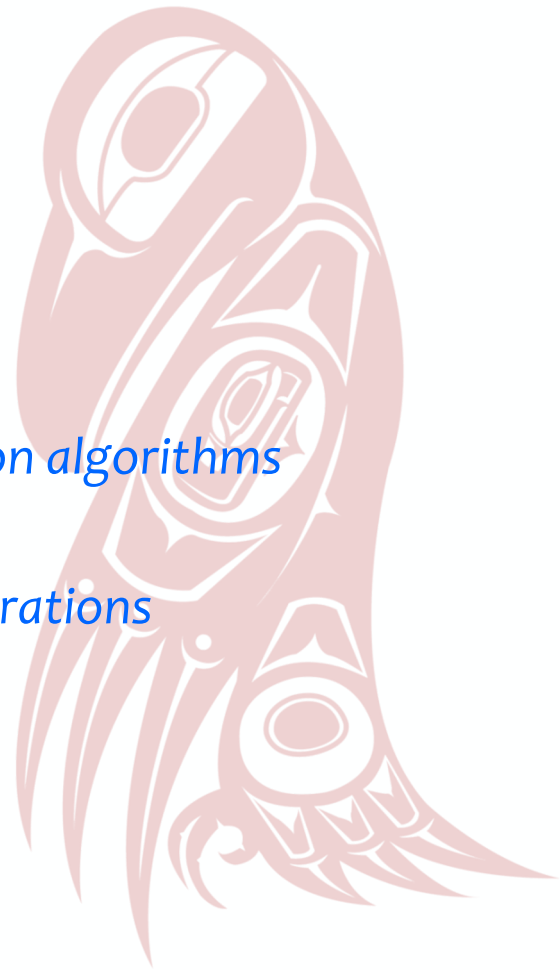
# Tradeoffs in Adaptive Streaming



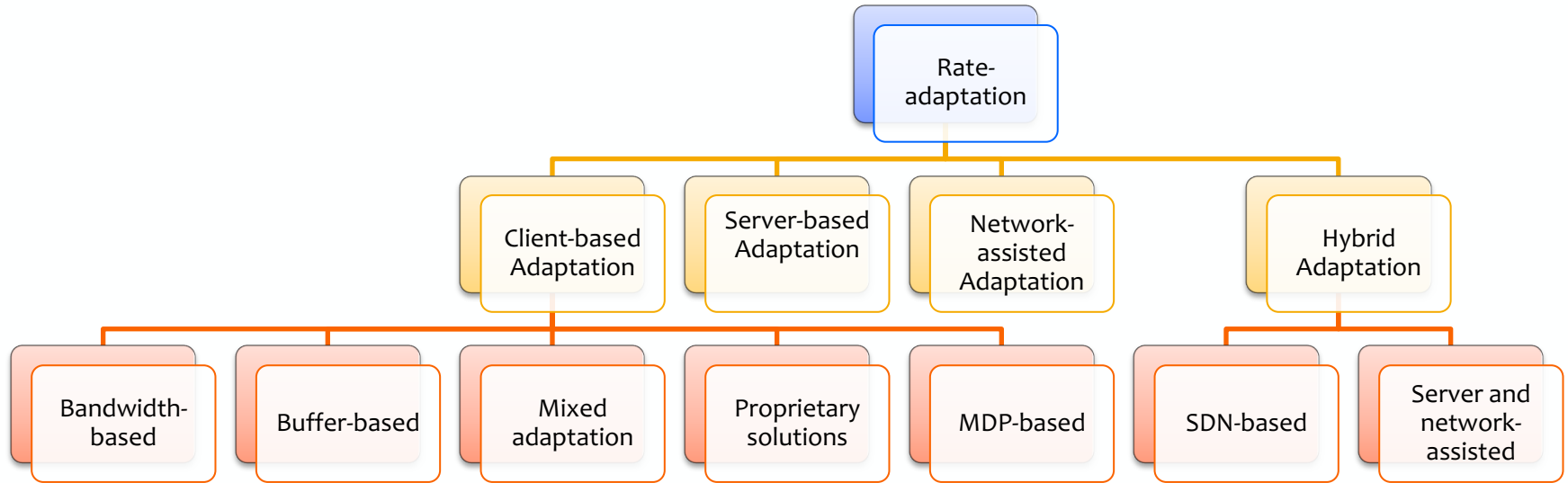


## Areas of Recent Progress

- *Improvements in client playback and rate-adaptation algorithms*
- *Low-latency live streaming extensions*
- *Content preparation for streaming: general considerations*
- *Per-title, content- and context-aware encoding*
- *Optimizations for multi-codec streaming*
- *Optimizations for multi-screen streaming*
- *HTTP streaming and CDN performance*



# Rate-Adaptation Algorithms



Reading: A survey on bitrate adaptation schemes for streaming media over HTTP – IEEE Commun. Surveys Tuts. (2019)

## Servers and clients cooperate

With each other and/or the network

😊: Using hints helps the clients and servers take more appropriate actions

## Server stay in control

Assume dumb clients

Apply appropriate QoS in the network

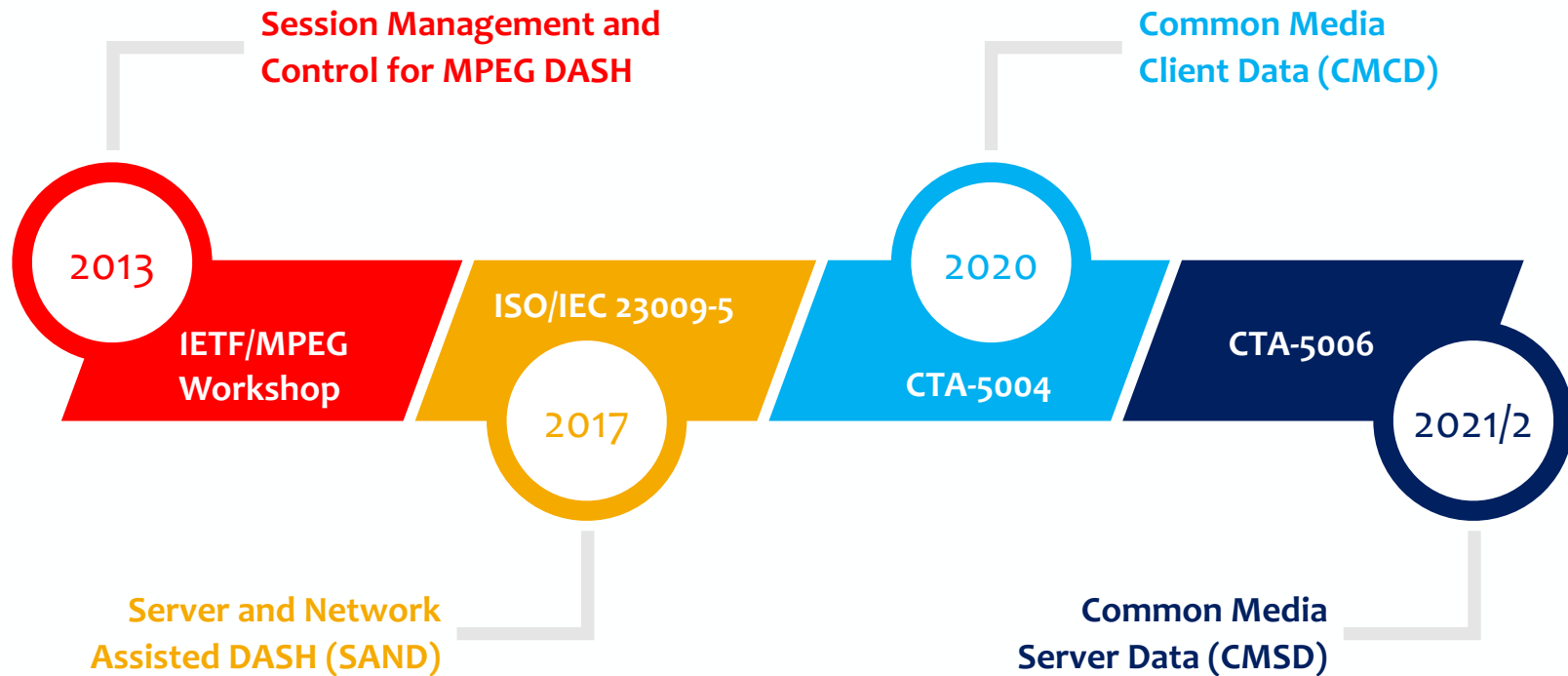
😊: Streaming video in uncontrolled fashion will never replace traditional video

## Clients stay in control

Assume dumb servers and network

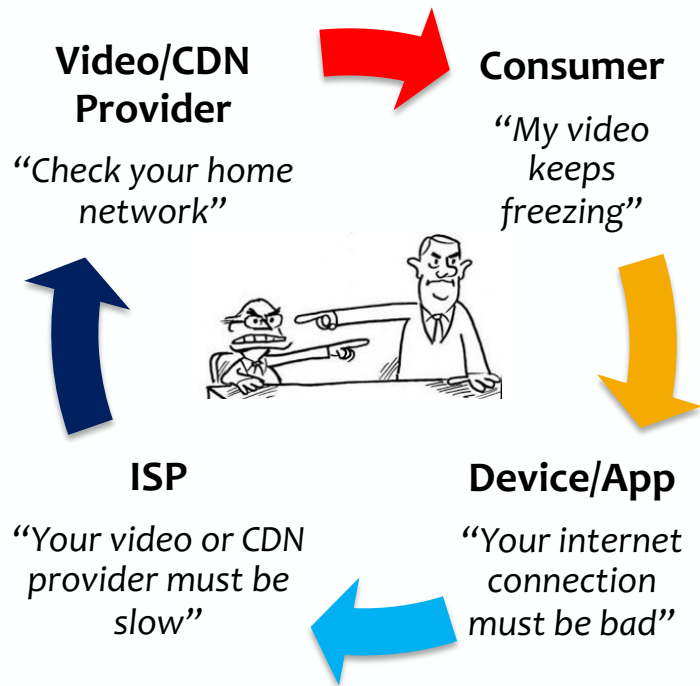
Be selfish

🤔: There is more bandwidth at the problem



# Whose Fault is It When My Video Sucks?

*Fault isolation requires analytics data from various points along the delivery pipeline*



Despite the common belief, CDNs

- are clueless about what they deliver
- cannot tie the individual GETs to playback sessions
- cannot generate dashboard metrics for
  - delivery performance
  - player software issues
  - viewer experience
- cannot prioritize delivery for urgent requests

## CTA-5004: Common Media Client Data (Published in Sept. 2020)

How could a client relay info about

- content ID and session ID
- current segment's type/duration/format
- delivery deadline
- next segment (or byte range) to be requested
- current buffer length, latency, startup delay and playback rate
- stall stats

Reading: Common media client data (CMCD): initial findings – ACM NOSSDAV'21

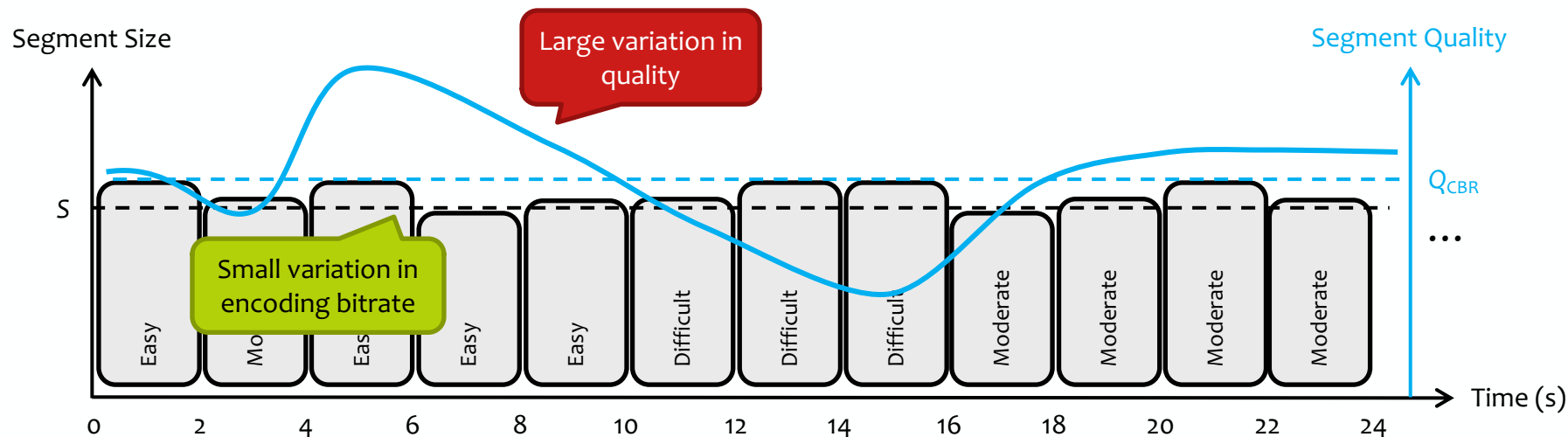
## CTA-5006: Common Media Server Data (Work started in May 2021)

How could a server relay info about

- server-side bandwidth estimates
- hints for the startup bitrate
- min/max limits for the playback bitrate
- redirection suggestions
- caching indications
- breadcrumb data
- server/network load signals

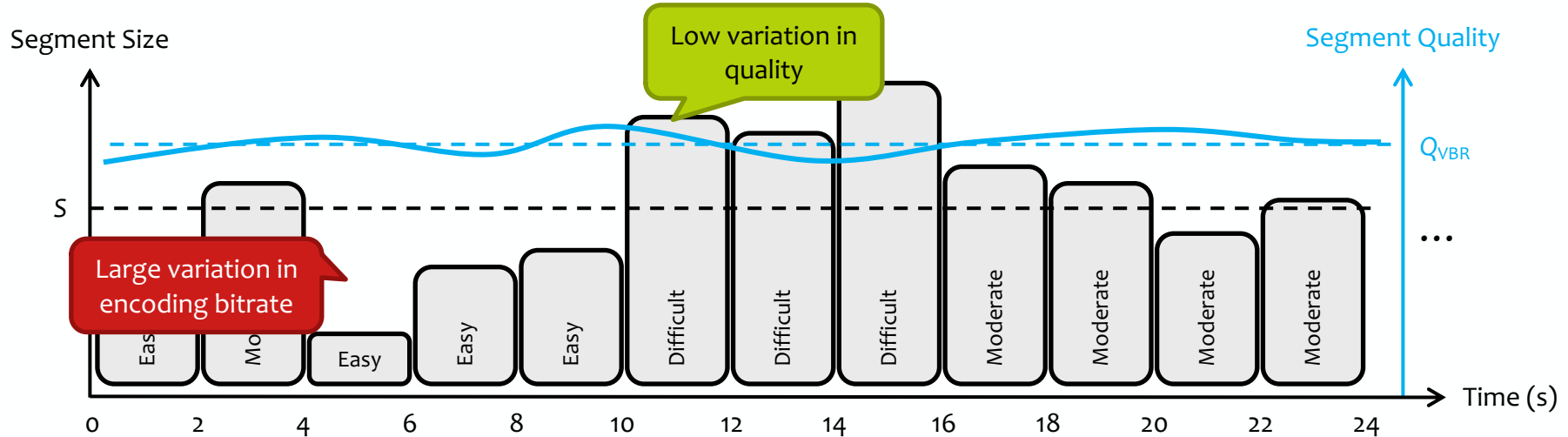
# Adaptation Feature Delivers Inconsistent Quality

*Guidelines limited bitrate variability to (mostly) 10% so far*



If there is something worse than having to watch a video at a lousy quality, it is to watch that video with varying quality

# What If We Encode in a More Subtle Fashion?

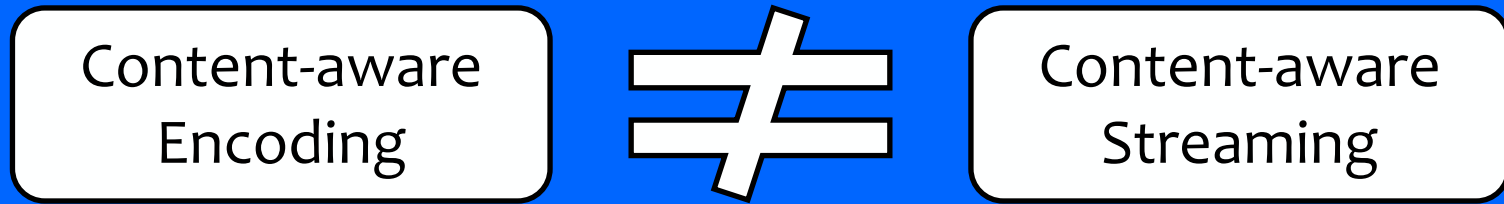


While we spend the same total amount of bits, we not only increase average quality but also reduce quality variation

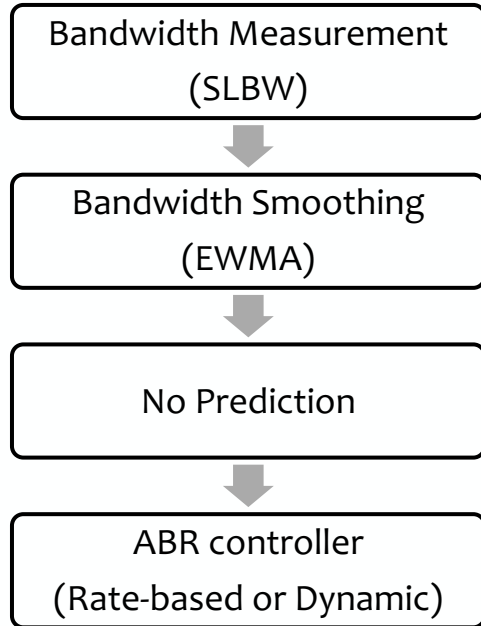
HLS authoring spec for ATV allows 2x capping rate for VoD. For linear content, variability is limited to 10-25% range.



**Generating content-aware-encoded segments is  
easy but streaming them is not!**



## Default dash.js

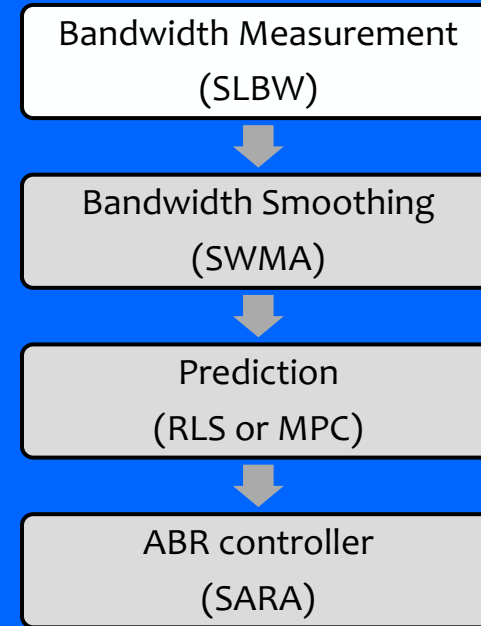


**SLBW:** Divides the segment size by download time

**EWMA:** (Exp. weighted) average of the last four segments

Reading: Adaptive streaming of content-aware-encoded videos in dash.js – IBC'21

## Size-Aware Rate Adaptation (SARA)

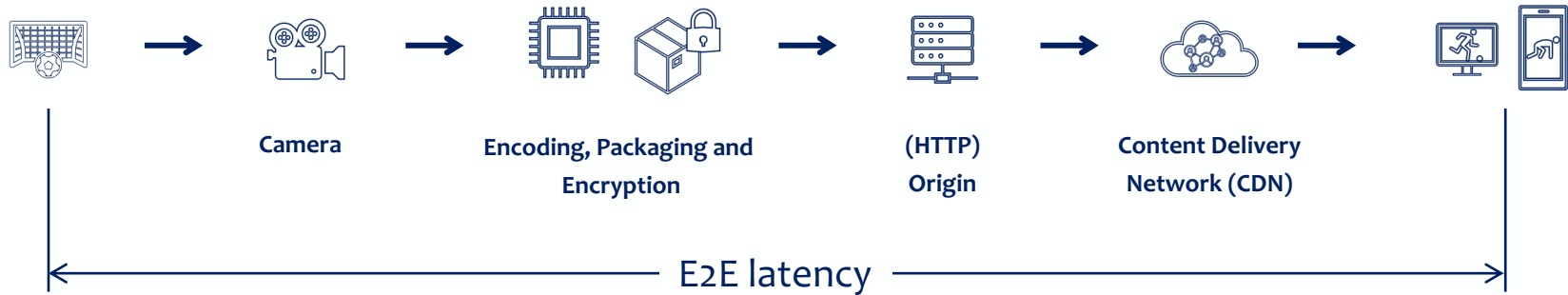


**SWMA:** Averages the last three segments

**RLS:** Recursive Least Squares

**MPC:** Model Predictive Control

# End-to-End (Glass-to-Glass) Latency



- Video encoding pipeline
- Ingest and packaging operations
- Network propagation
- Server I/O, CDN buffering
- Media segment duration
- License/key fetching, decryption
- Player behavior
  - Buffering
  - Playhead positioning
  - Resilience

High latency

Typical latency

Low latency

Real time

DASH/HLS  
10s segments

DASH/HLS  
6s segments

DASH/HLS  
2s segments

Live sports, game streaming, eSports

Social media, messaging

Cable, IPTV, satellite, OTA

DASH/HLS  
1s segments

DASH/HLS  
chunked segments

VoIP

Gambling, auctions

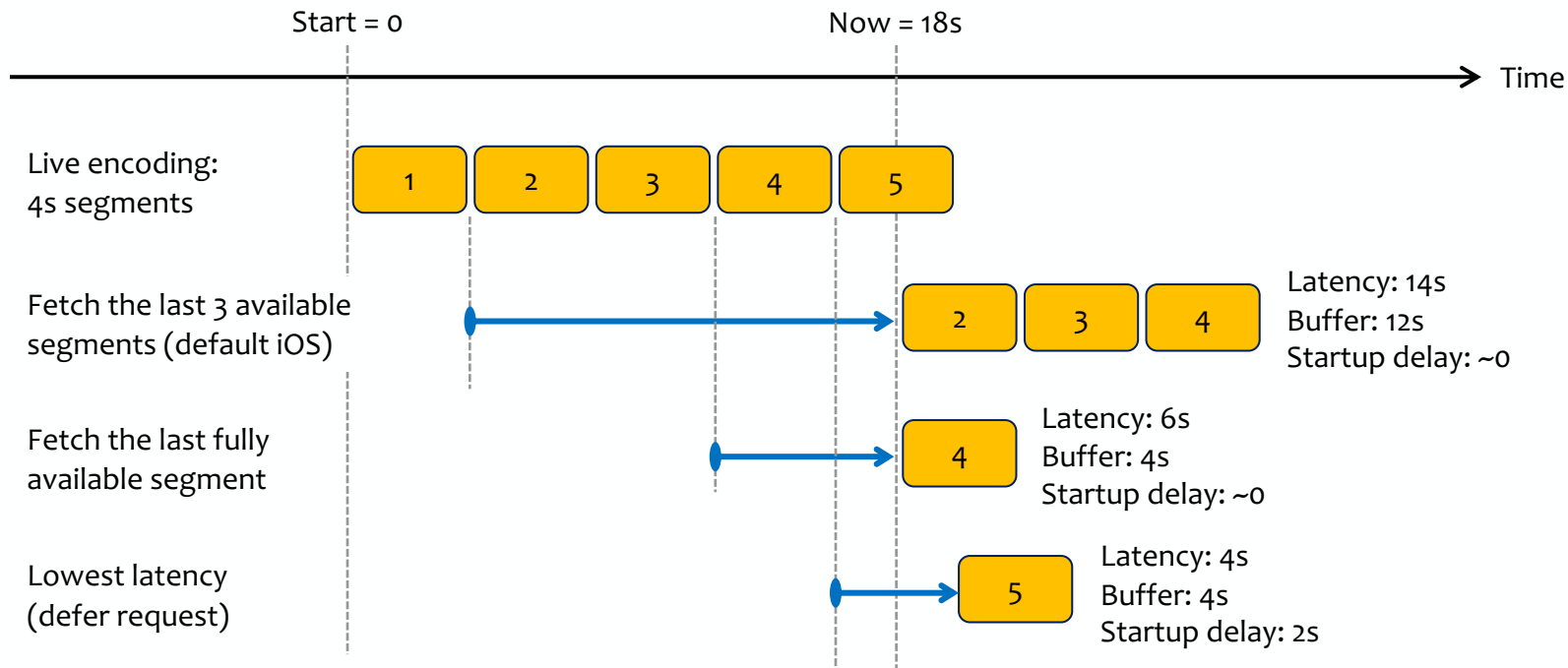
45  
seconds

10  
seconds

1  
second

# Segmented Content Example

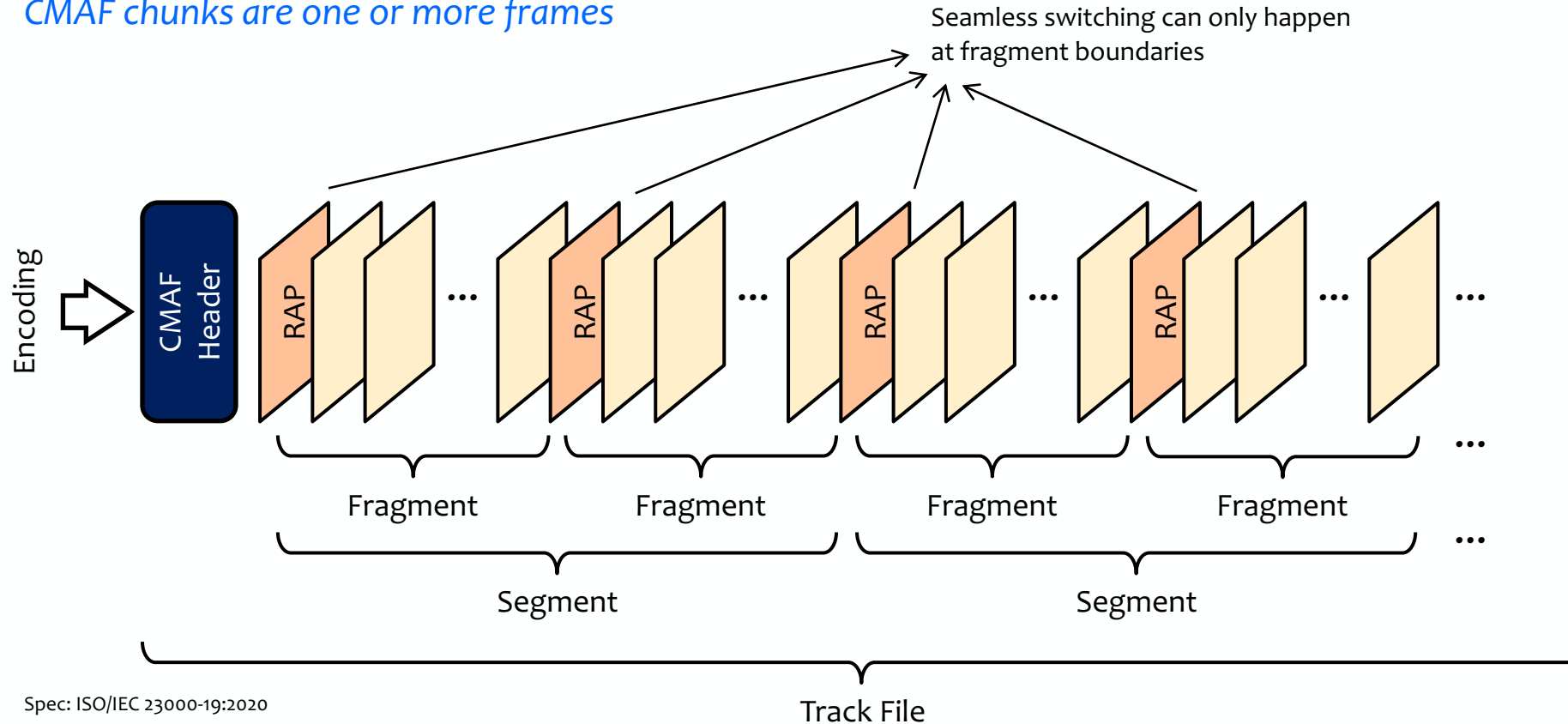
*Startup delay  $\neq$  Latency*



\* Segment fetching time is assumed to be negligible in this example

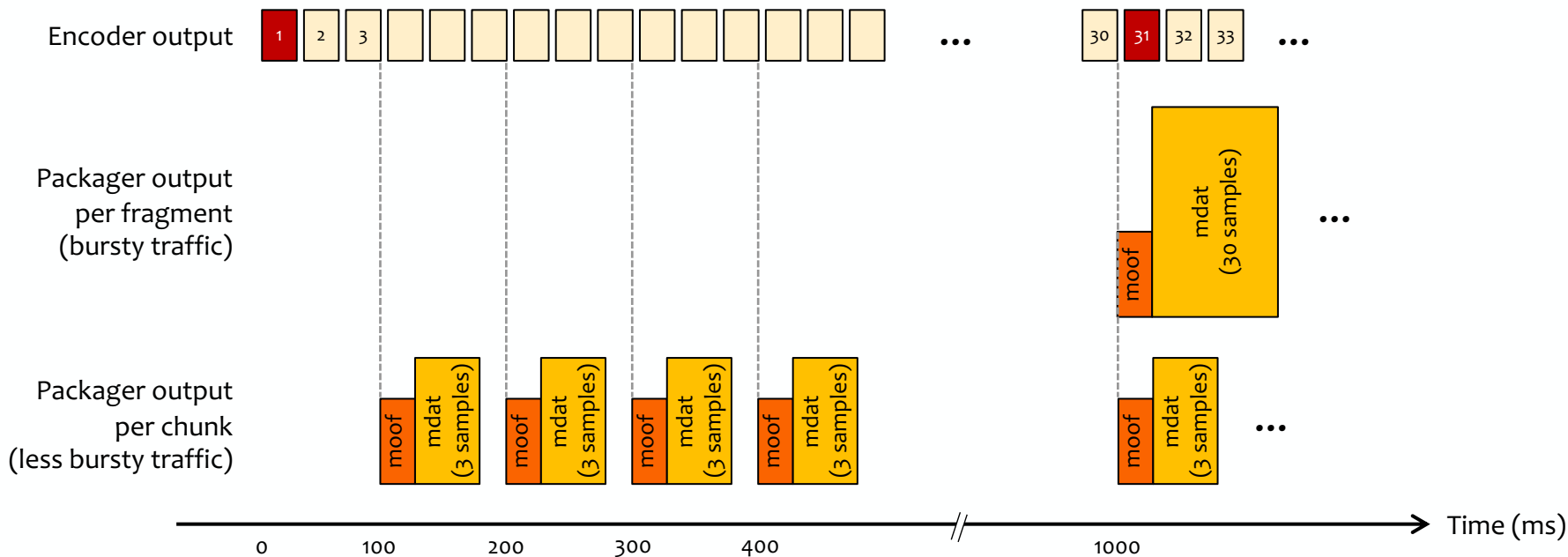
# Chunking Enabled by Common Media Application Format (CMAF)

*CMAF chunks are one or more frames*



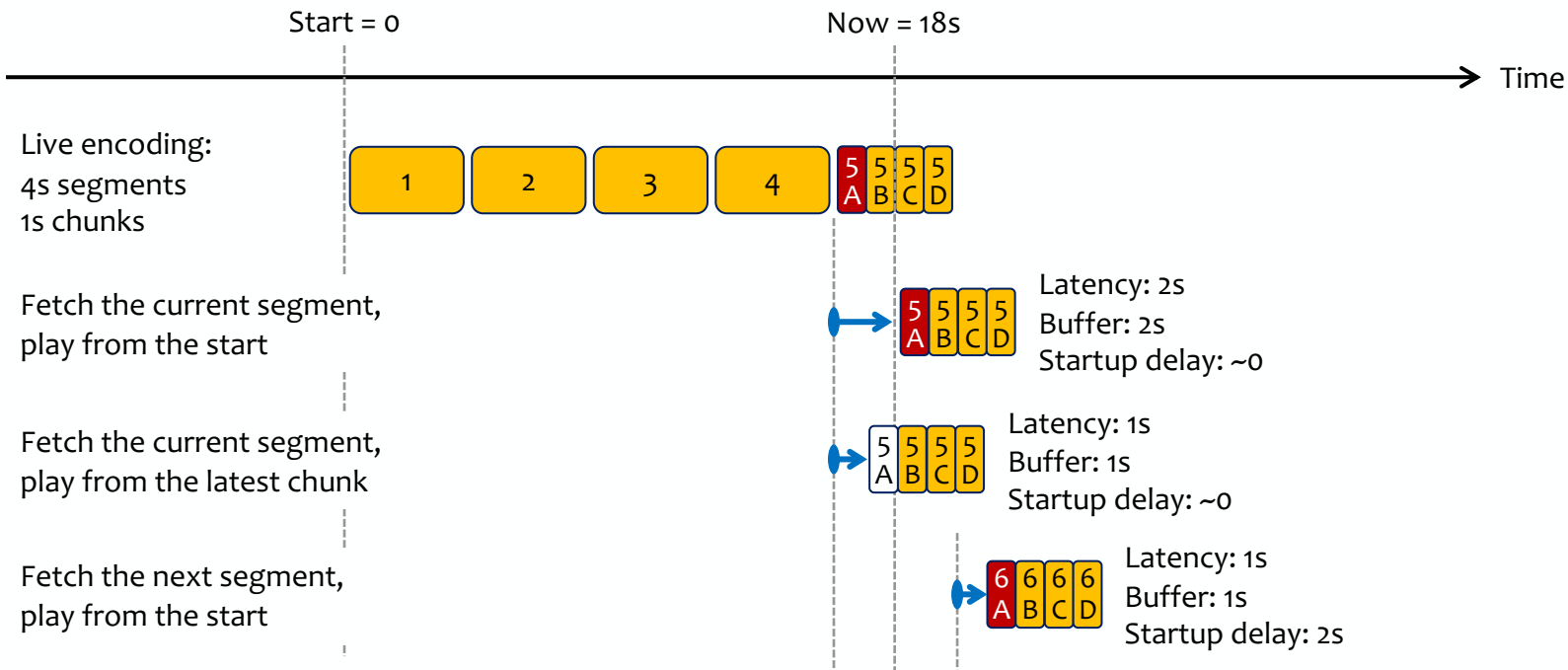
# Low Latency Enabled by CMAF Chunks

*Example: CMAF fragment containing a 30-fps video of 30 samples*



# Chunked Content Example

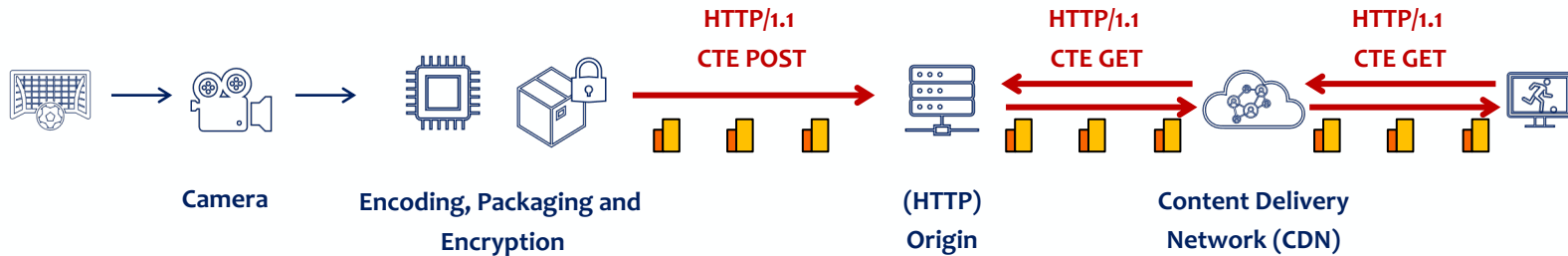
*Chunking decouples latency from segment duration*

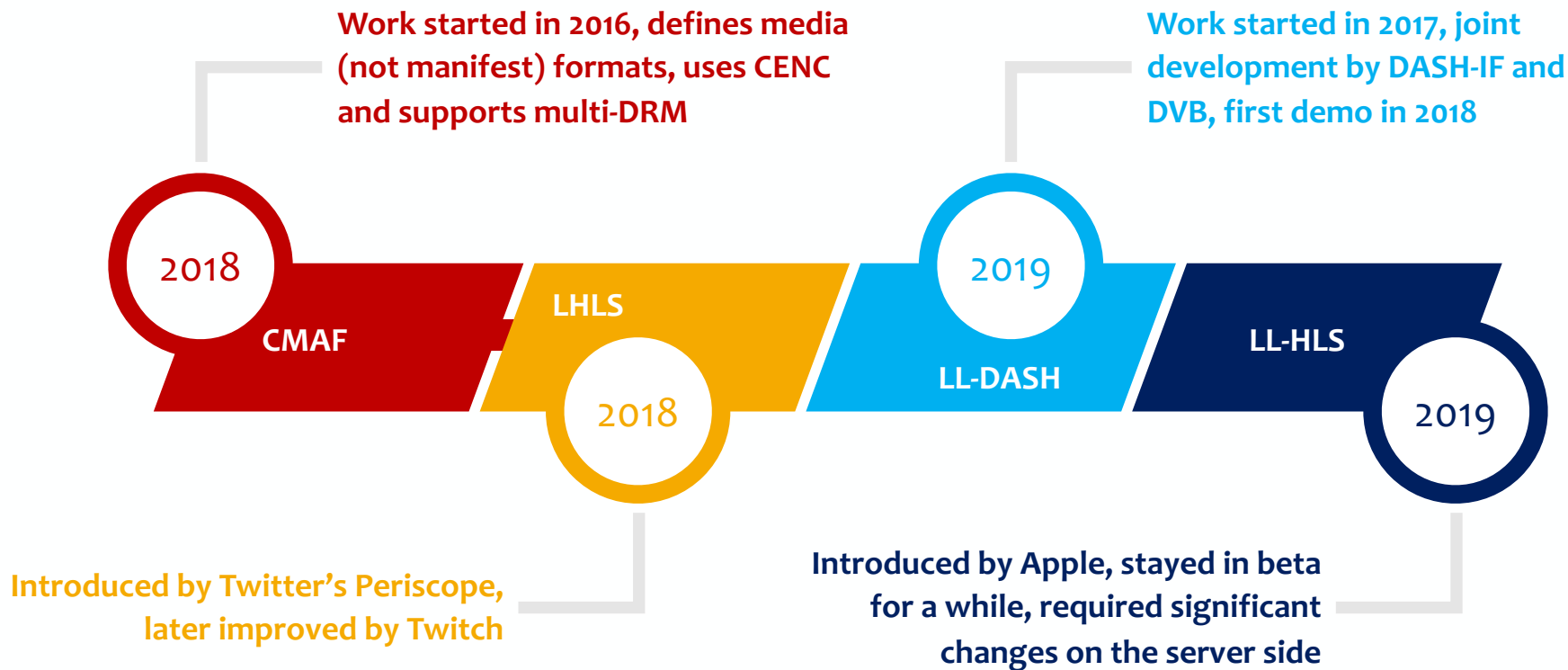


\* Chunk fetching time is assumed to be negligible in this example



# Low Latency Enabled by Chunked Transfer Encoding (CTE)





# Industry Efforts for Low Latency

	Chunked Content	Use of CTE	Describe Segment Structure	Per-chunk Manifest Refresh	Line-speed Delivery	Smart Origin to Modify Manifest	Deterministic Startup
LHLS	✓	✓	✗	✗	✗	✗	✗
LL-DASH	✓	✓	✓ *	✗	✗	✗	✗
LL-HLS	✓	✗	✓	✓	✓	✓	✓

\* Certain ways to describe segment structures for DASH are to be published in DASH 5<sup>th</sup> edition

# LHLS: Community's Solution to High-Latency HLS

<https://github.com/video-dev/hlsjs-rfcs/pull/1/commits>

#EXTM3U

#EXT-X-VERSION:3

#EXT-X-TARGETDURATION:2

#EXT-X-PROGRAM-DATE-TIME:2018-09-05T20:59:06.531Z

#EXTINF:2.000 <https://foo.com/bar/0.ts>

#EXT-X-PROGRAM-DATE-TIME:2018-09-05T20:59:08.531Z

#EXTINF:2.000 <https://foo.com/bar/1.ts>

#EXT-X-PREFETCH:<https://foo.com/bar/2.ts>

#EXT-X-PREFETCH:<https://foo.com/bar/3.ts>

# LL-DASH: Low-Latency Extensions for DASH

<https://dashif.org/news/low-latency-dash/>

- Low-latency service offering
- Chunked and on-chunked low-latency adaption sets
- Delta with DVB's version
- Informative client guidelines for operation
- Recommendations for encoding and chunking
- Latency calculations and use of prtf
- Resynchronization points
- Service configuration parameters for low latency
- Example FFmpeg configuration, MPDs

CHANGE REQUEST			
DASH-IF IOP	CR	rev -	Current version: V4.3
Status:	<input type="checkbox"/> Draft	<input type="checkbox"/> Community Review	<input type="checkbox"/> Agreed in TF <input checked="" type="checkbox"/> Agreed <input type="checkbox"/> Stable
Title:	Low-latency Modes for DASH		
Source:	Live TF		
Supporting Companies:	Akamai, Amazon Elemental, castLabs, Comcast, Elemental Technologies, Ericsson, Frontier Communications Harmonic, Hulu, Qualcomm Incorporated, Sony, TNO, Unified Streaming,		
Category:	<b>A</b> Use <u>one</u> of the following categories: <b>C</b> (correction) <b>A</b> (addition of feature) <b>B</b> (editorial modification)	Date: 2020-03-27	
Reason for change:	DASH-IF collected information related to Low-Latency Streaming in a Report, together with DVB. The report (available here: <a href="https://dash-industry-forum.github.io/docs/Report%20on%20Low%20Latency%20DASH.pdf">https://dash-industry-forum.github.io/docs/Report%20on%20Low%20Latency%20DASH.pdf</a> ) provides use cases, service scenarios, deployment experience and existing potential technologies. Also, DASH-IF already generated the DASH profile for ATSC that includes a mode supporting low latency.  Based on this information in the report, it is recommended to add low latency to DASH-IF IOP Guidelines.		
Summary of change:	This change provides a new clause for live services that addresses specification updates as well as implementation guidelines to support Low-Latency DASH services addressing the requirements above.		
Consequences if not approved:	Low-Latency DASH not available		
Sections affected:	References in clause 2 New section 9.X in v5 of DASH-IF IOP		

# Noteworthy Items from LL-DASH

- Two operational modes are permitted
  - Simple live offering with @duration signaling and \$Number\$ based templating
  - Main live offering with SegmentTimeline (\$Number\$ or \$Time\$)
- At least one ServiceDescription element shall be present as follows:
  - A Latency element: Min, target and max latencies
  - A PlaybackRate element: Min and max playback speeds for latency correction
- At least one UTCTiming element with millisecond precision
- A low latency adaption set can be built in two ways
  - Using non-chunked segments whose duration < 30% of target latency
  - Using chunked segments

# Resynchronization Points

- Introduced in DASH 5th edition (FDIS)
  - Defines the location of intermediate resynchronization points within media segments
- Benefits are
  - Fast random access while maintaining low latency
  - Quick resynchronization after buffer underruns
  - In-segment downshifting in case of a stall risk
  - Understanding the applied chunk size and duration (support for the rate adaptation)
- How to find them
  - By providing a binary map for each resynchronization point in a resynchronization index segment
    - This is used for segments that are fully available, so not very useful for low latency
  - By signaling in the MPD the existence of resynchronization points (the client can parse the segment to locate the resynchronization points)

# Resynchronization Points

## Examples

```
<Resync type="0" dT="500000" dImin="0.03125" dImax="0.09375"/>  
<SegmentTemplate timescale="1000000" .../>  
<Representation id="0" mimeType="video/mp4" bandwidth="500000" .../>
```

- Indicates that
  - The point is of type 0: a CMAF chunk without any promise for specific random-access capabilities
  - The maximum chunk duration is 0.5s (500,000/1,000,000)
  - The min/max distance in bytes between two resynchronization points:  $0.03125 \times 500,000 - 0.09375 \times 500,000$

```
<Representation id="2" mimeType="video/mp4" bandwidth="300000" ...>  
<Resync type="2" dT="1000000" dImin="0.1" dImax="0.15" marker="TRUE"/>  
</Representation>
```

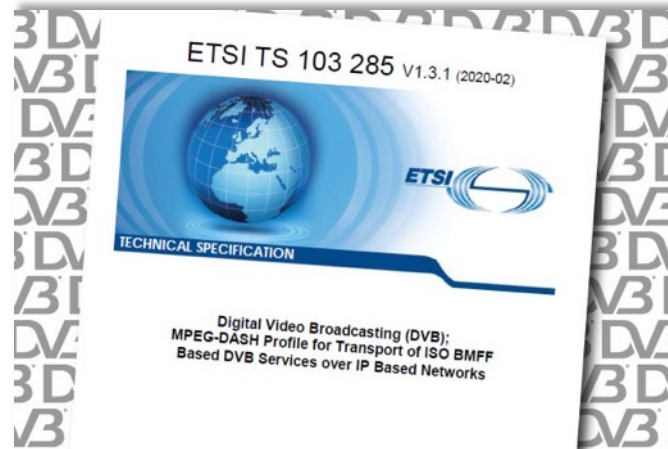
- Indicates that
  - The point is of type 2: a CMAF chunk that can be used for fast access or switching
  - The maximum time delta between these points is 1s (1,000,000/1,000,000)
  - The min/max distance in bytes between two resynchronization points:  $0.1 \times 300,000 - 0.15 \times 300,000$
  - As the @marker flag is set to true, a client may search for the resynchronization point using a box-parsing algorithm



# DVB-DASH Low Latency

## TS 103 285 V1.3.1

- Free to download from
  - <https://dvb.org/?standard=dvb-mpeg-dash-profile-for-transport-of-iso-bmff-based-dvb-services-over-ip-based-networks>



# LL-DASH Capabilities in dash.js

- **Description of the parameters**
  - <https://reference.dashif.org/dash.js/nightly/samples/low-latency/index.html>
- **Adjusting low-latency attributes and parameters**
  - <https://reference.dashif.org/dash.js/nightly/samples/low-latency/testplayer/testplayer.html>
- **dash.js Wiki for low latency**
  - <https://github.com/Dash-Industry-Forum/dash.js/wiki/Low-Latency-streaming>

## Coming next

- Support for resynchronization points
- Improved low-latency scheduling logic
- Improve ABR algorithms
  - L2A
  - LoL+
  - AAST-based approach
- Support for prft boxes

# LL-HLS: Low-Latency Extensions for HLS

[https://developer.apple.com/documentation/http\\_live\\_streaming/enabling\\_low-latency\\_hls](https://developer.apple.com/documentation/http_live_streaming/enabling_low-latency_hls)

- Brings production latency down to few seconds
- Backward compatible with older HLS clients by still listing the full segments in the playlist
- Preserves characteristics of HLS
  - Playlist update/load for each (partial) segment
  - Rate adaptive
  - Encryption, ads and metadata support
- Encodes segments in chunks (support for mp4/CMAF and TS)
  - Chunks are called parts, each has a unique URL (cacheable)
  - Not every part contains an IDR (the INDEPENDENT attribute)
- Imposes new requirements on the origin
  - Support for new tags and H2

# Apple Introduced Several Concepts in LL-HLS



## Partial segments (parts)

Packaged, published and added to the playlist earlier than the parent segment



## Playlist delta updates

Allows the clients to only request the new portions the playlist, reduces overhead



## Blocking of playlist reload

Avoids constant polling of the servers, responses are held till the specified future segment is available



## Rendition reports

Each media playlist contains information about other renditions to allow faster switching



## Preload hints

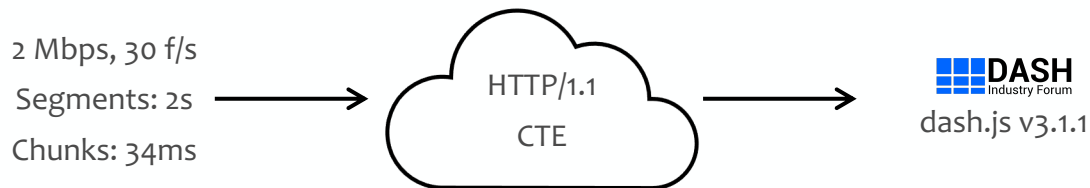
Informs the client about the upcoming parts

```

#EXTM3U
#EXT-X-TARGETDURATION:4
#EXT-X-VERSION:6 #EXT-X-SERVER-CONTROL:CAN-BLOCK-RELOAD=YES,CAN-SKIP-UNTIL=24,PART-HOLD-BACK=3.012
#EXT-X-PART-INF:PART-TARGET=1.004000
#EXT-X-MEDIA-SEQUENCE:236928
#EXT-X-MAP:URI="fileSequence56.mp4"
#EXT-X-PROGRAM-DATE-TIME:2021-02-23T16:34:54.261Z
#EXTINF:4.00000,
fileSequence237476.mp4
#EXTINF:4.00000,
fileSequence237477.mp4
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237478.1.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237478.2.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237478.3.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237478.4.mp4"
#EXTINF:4.00000,
fileSequence237478.mp4
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237479.1.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237479.2.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237479.3.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237479.4.mp4"
#EXTINF:4.00000,
fileSequence237479.mp4
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237480.1.mp4"
#EXT-X-PART:DURATION=1.00000,INDEPENDENT=YES,URI="lowLatencySeg.mp4?segment=filePart237480.2.mp4"
#EXT-X-PRELOAD-HINT:TYPE=PART,URI="lowLatencySeg.mp4?segment=filePart237480.3.mp4"
#EXT-X-RENDITION-REPORT:URI="/cmf/aud/lowLatencyHLS.m3u8",LAST-MSN=237578,LAST-PART=0
#EXT-X-RENDITION-REPORT:URI="/cmf/media0/lowLatencyHLS.m3u8",LAST-MSN=236944,LAST-PART=1
#EXT-X-RENDITION-REPORT:URI="/cmf/media2/lowLatencyHLS.m3u8",LAST-MSN=236944,LAST-PART=0

```

# Single-Client Traffic Comparison for LL-DASH vs. LL-HLS



In 60 seconds:  
- 30 requests for 30 segments  
~15 MB media download



In 60 seconds:  
- 1800 requests for parts  
~15 MB media download  
- 160 requests for playlist updates  
~150 KB non-media download

# Summing up LL-HLS

## Observations

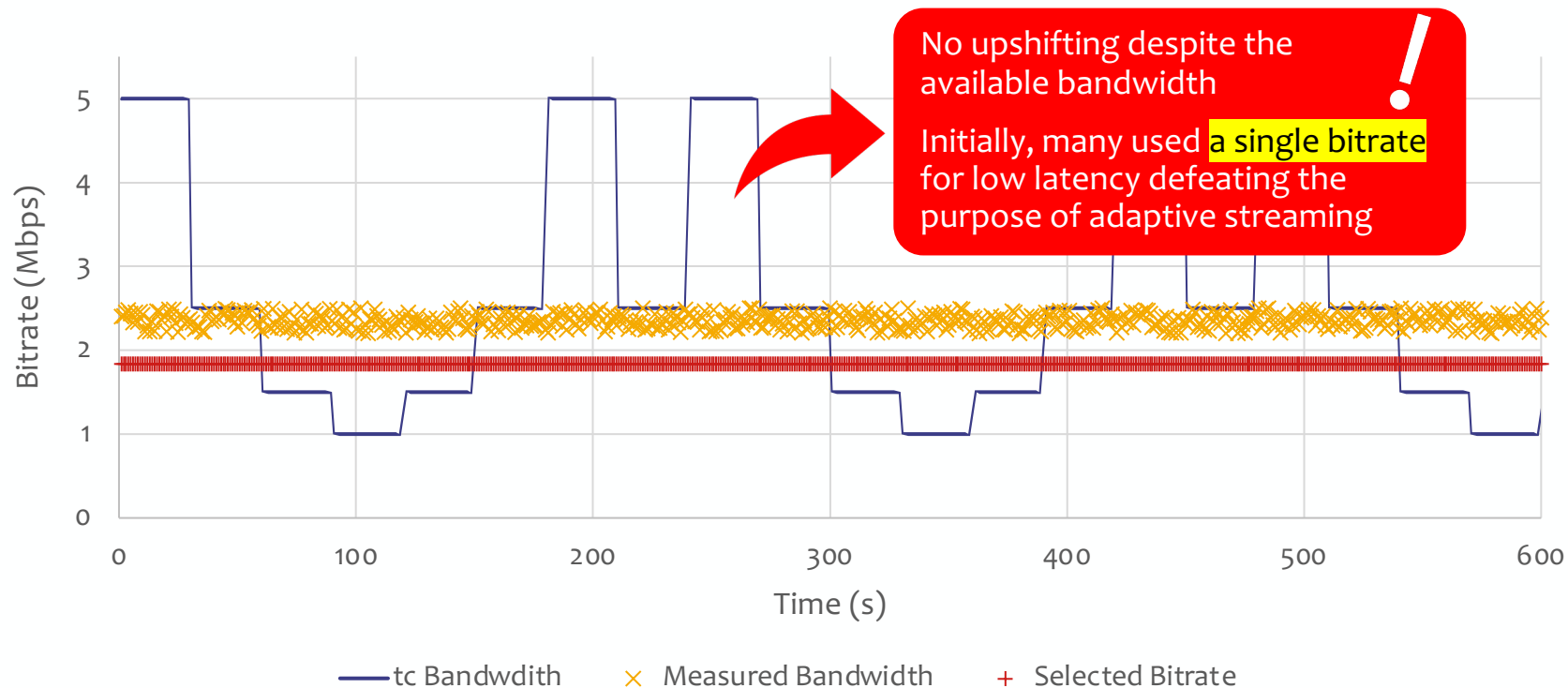
- LL-HLS performance relies on selecting the appropriate settings
  - Lowest latency with parts of 300ms or shorter
  - Startup delay increases with the rendition bitrate
  - Both latency and startup delay become less stable in worse network conditions
- LL-HLS transmits the parts at the full line speed
  - Implications on bandwidth measurements and rate adaptation

## Concerns

- With too short segments and parts
  - LL-HLS incurs non-media overhead
  - LL-HLS makes vastly more requests
- LL-HLS may consume significantly more origin and cache resources
  - Impact on the scalability

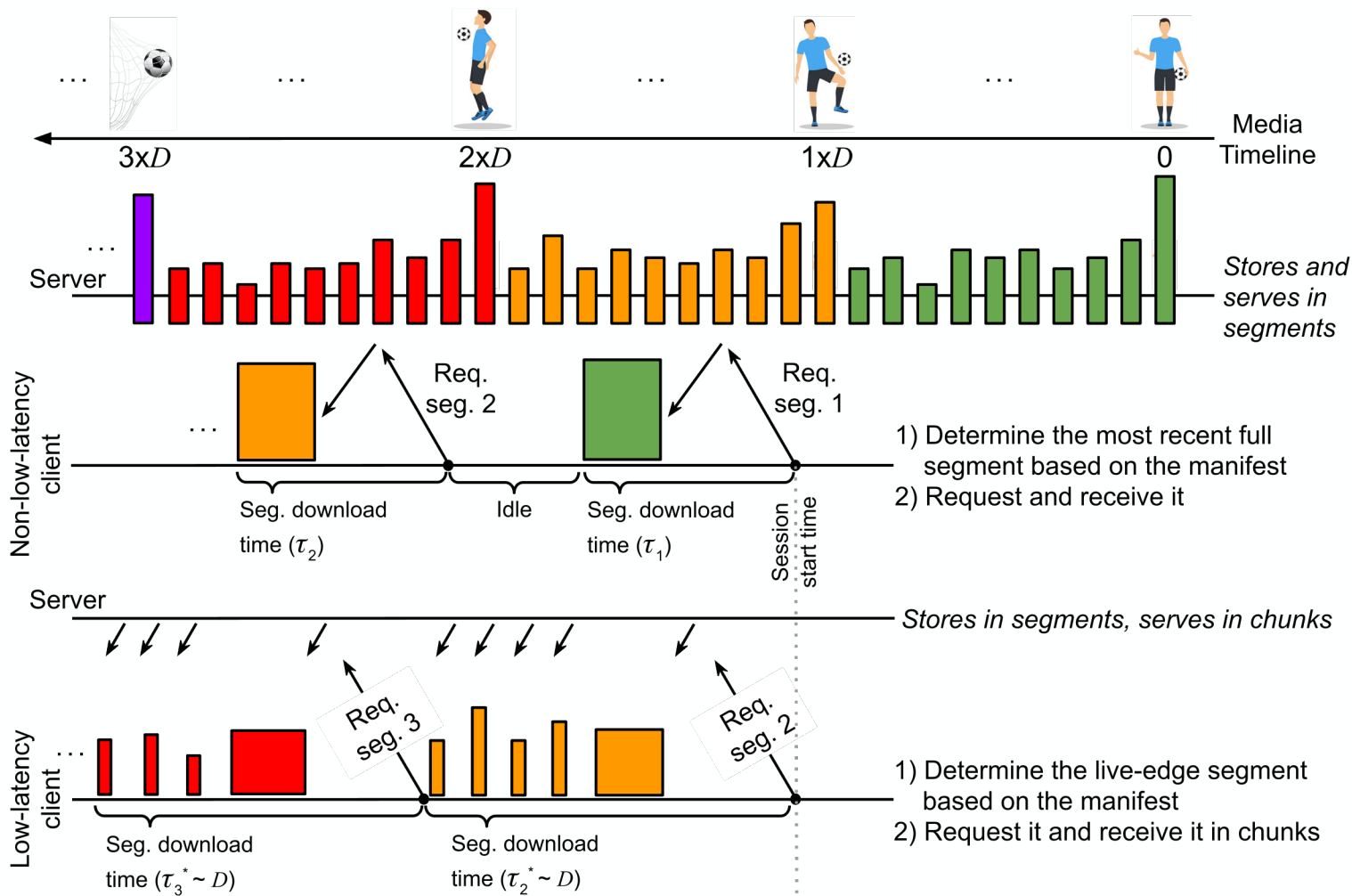
# Bandwidth Measurement is Tricky

Live Twitch data (Nov. 2018)

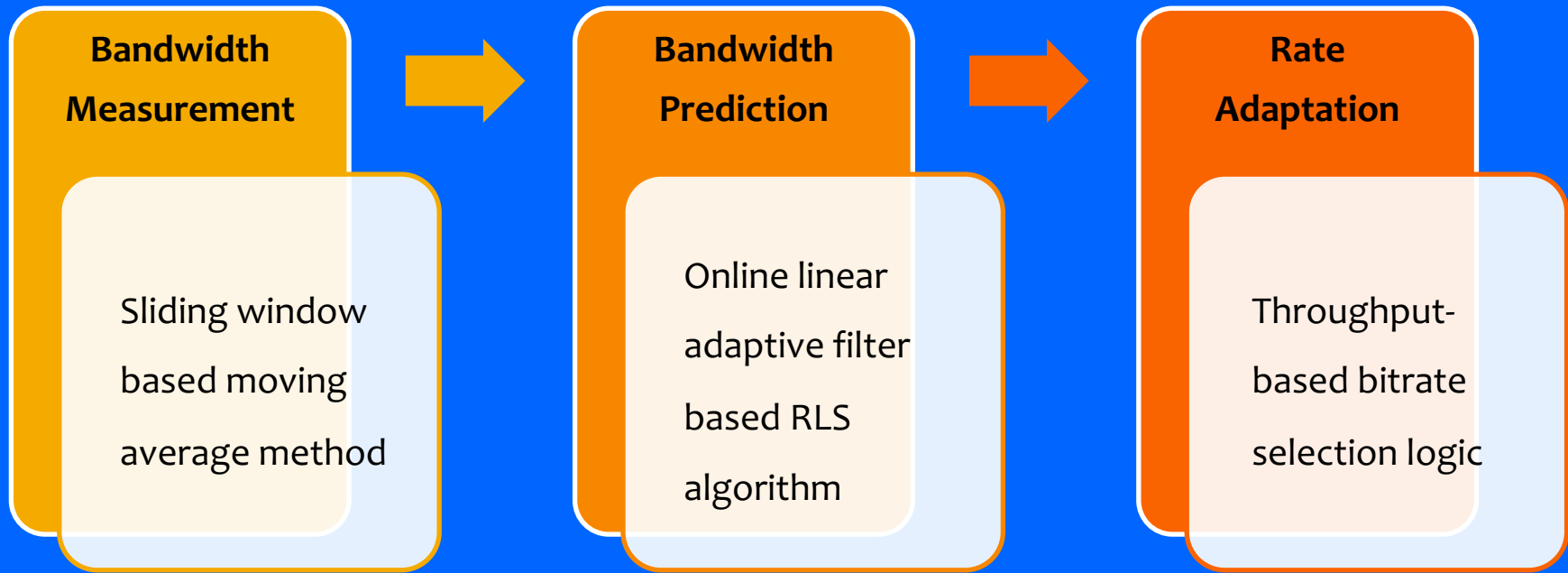


\* Encoded at {0.18, 0.73, 1.83, 2.5, 3.1, 8.8} Mbps with three resolutions of {540p, 720p, 1080p} and packaged with CMAF





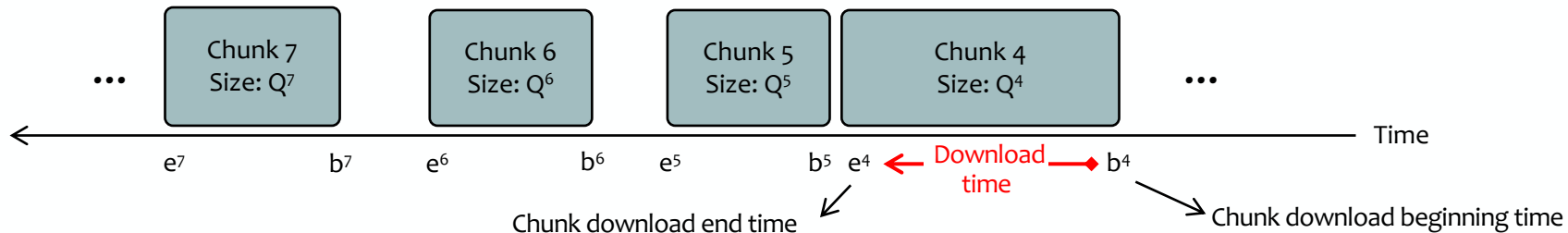
# ABR for Chunked Transfer Encoding (ACTE)



Reading: Bandwidth prediction in low-latency chunked streaming – ACM NOSSDAV'19

More reading: Performance analysis of ACTE: a bandwidth prediction method for low-latency chunked streaming – ACM TOMM (2020)

# Bandwidth Measurement



- A chunk-based algorithm improving ACTE:
  - Chunk boundary identification
    - Computes the beginning and end time of each chunk download by capturing the ‘moof’ box
  - Chunk filtering
    - Ignores the first and the last chunks to avoid any transient outliers
- Fetch API allows
  - Tracking the progress of chunk downloads
  - Parsing the chunk payloads in real time
- So, we can
  - Store the chunk beginning time when a ‘moof’ box of a chunk is captured
  - Store the chunk end time and size when the chunk is fully downloaded

Reading: Data-driven bandwidth prediction models and automated model selection for low latency – IEEE TMM (2021)

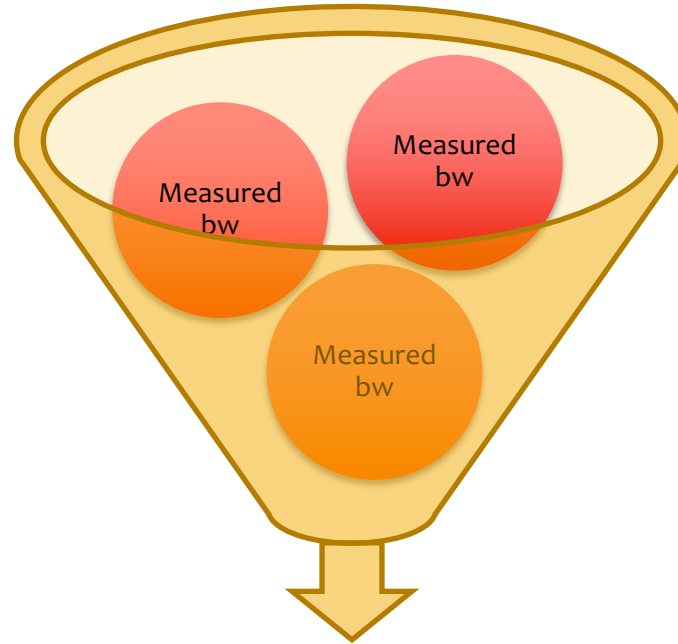
# Bandwidth Measurement

## *Identifying the “good” chunks without the $b^n$ values*

- Reasonable assumption: Idle periods cannot happen within a chunk, happen only between the chunks
  - Since the server pushes the chunks at full network speed
- For each chunk, compute its download rate, which equals its size divided by this chunk's end time minus previous chunk's end time
  - If this download rate is close ( $\pm 20\%$ ) to the average segment download rate, there must be significant idle time between these two chunks
    - Transmission is source limited
    - Disregard the current chunk
  - Else, the idle time is negligible
    - Transmission is network limited
    - The current chunk's download rate is a good approximation of the available bandwidth
- Use a sliding window based moving average method over the last three successful chunk downloads

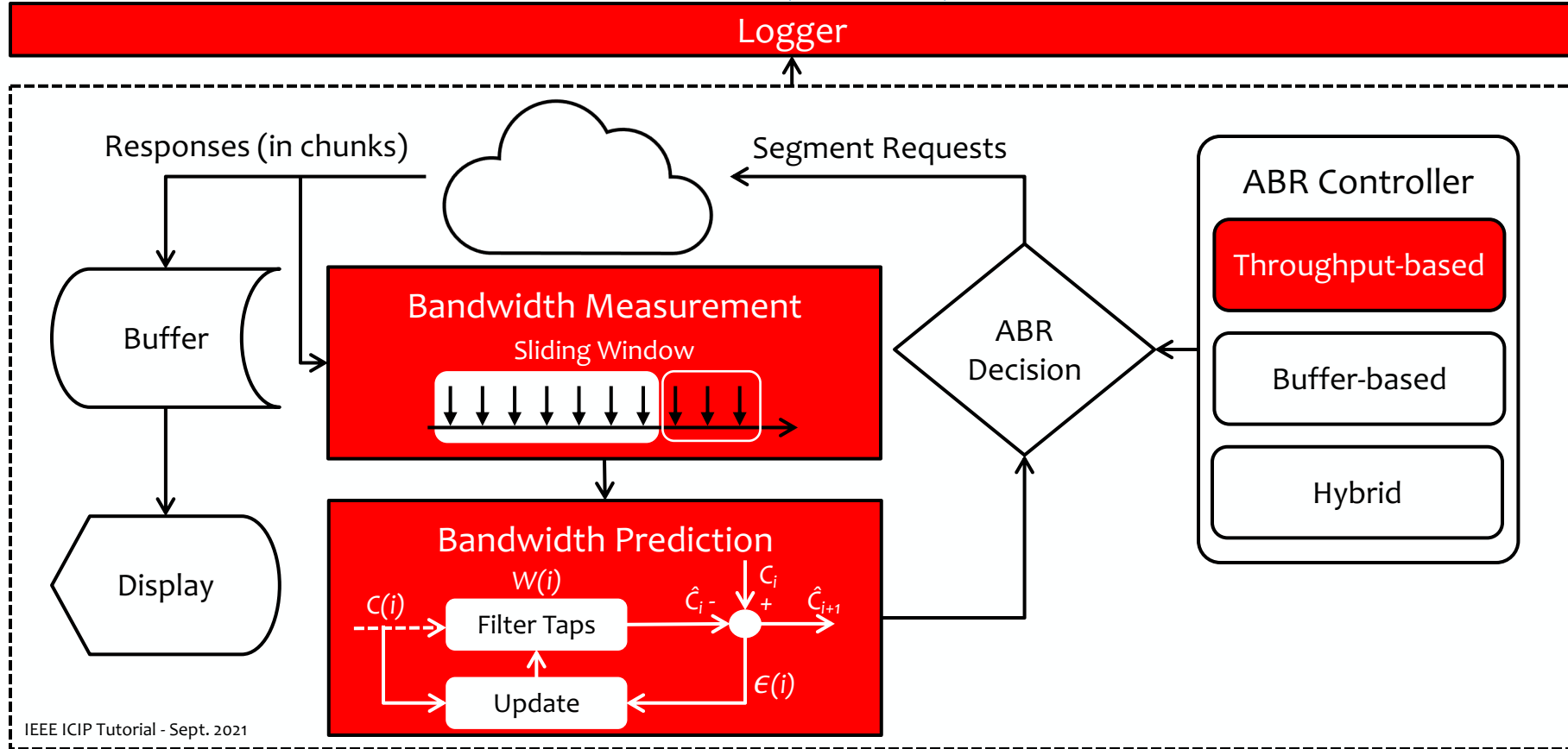
# (Future) Bandwidth Prediction and Rate Adaptation

*Online linear adaptive filter using recursive least squares (RLS)*



**Predicted bandwidth  
and rate adaptation**

## New/Modified Blocks in dash.js (in Red)



# Adaptation Algorithms for Near-Second Latency

Grand Challenge Organized and Sponsored by



**Submissions:** Apr. 3, 2020

**Presentations:** @ ACM MMSys'20 week

**Twitch's blog post and testbed:** <https://tinyurl.com/2020llgc>

**Awards:** Winner (5,000 USD) and runner-up (2,500 USD)

## Goals

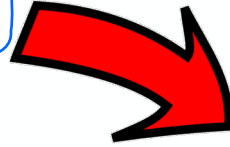
Keep the latency around one second

Balance the factors to maximize viewer experience

<https://www.youtube.com/watch?v=rcXFVDotpy4>

## ABR for CTE (ACTE)

(published in ACM NOSSDAV'19)



## Low-on-Latency (LOL)

(published in ACM MMSys'20, awarded by Twitch)



**Grand Challenge on Adaptation Algorithms for Near-Second Latency**

**Runners-up**  
May Lim, Mehmet N. Akcay, Abdelhak Bentaieb,  
Ali C. Begen and Roger Zimmermann

The 11<sup>th</sup> day of June 2020  
John Bartos



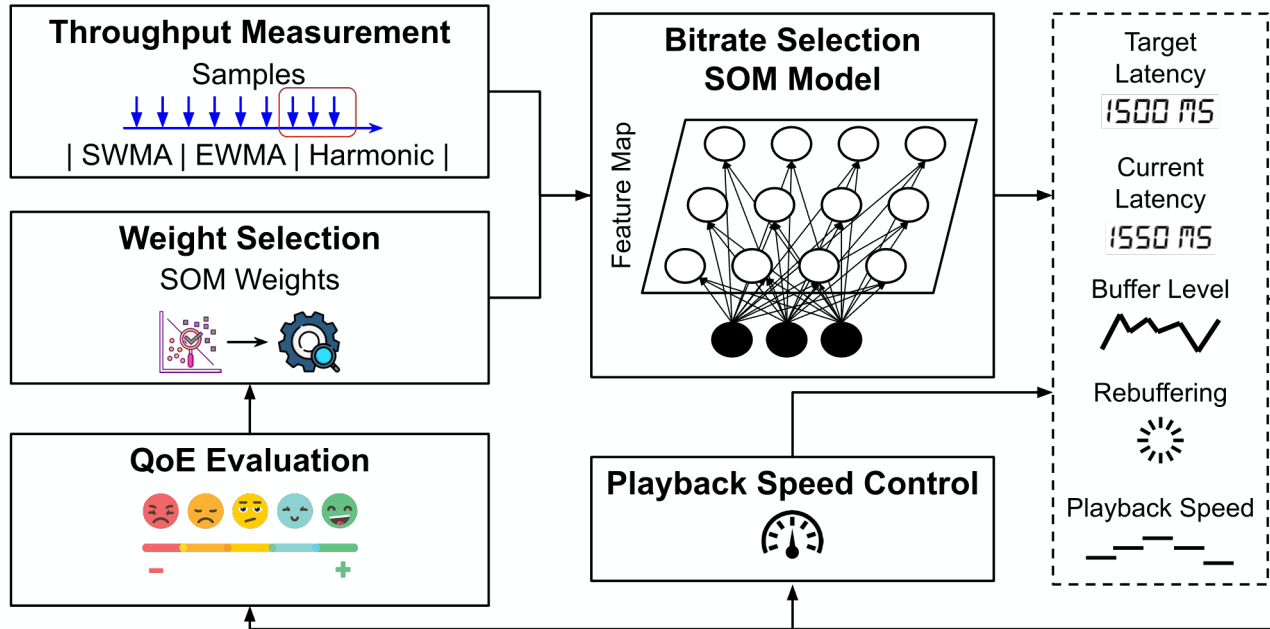
## LOL+

(to appear in IEEE TMM)



# Improvements from LoL to LoL+

## Automated weight selection

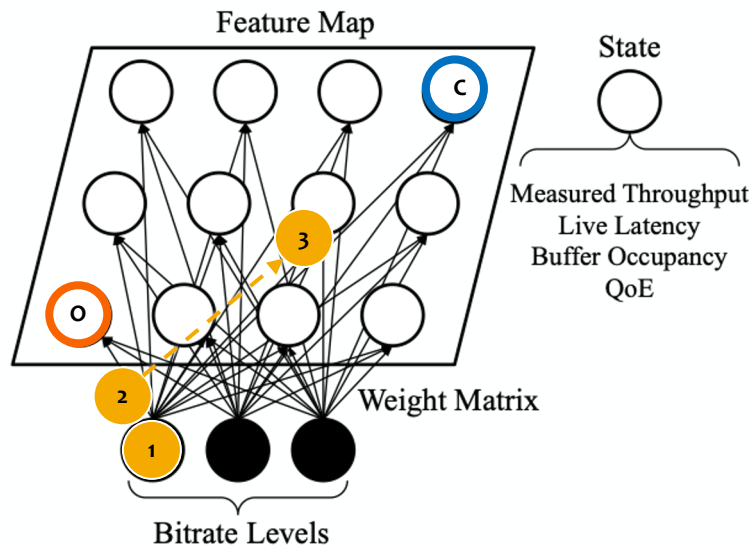


Reading: Catching the moment with LoL+ in Twitch-like low-latency live streaming platforms – IEEE TMM (2021)

Code: <https://github.com/NUStreaming/LoL-plus>

# Learning-Based ABR Algorithm

## Based on Self-Organizing Maps (SOM)



- Select the neuron closest to the optimal state
  - The optimal state consists of target latency, buffer and QoE values
  - Selected neuron gives the next bitrate to request
- Update selected neuron to move closer to the optimal state
- After segment download, update selected neuron to move closer to its consequent state
  - The consequent state consists of the resultant latency, buffer and QoE values

\* Each bitrate level translates to one neuron

# Playback Speed Control Module

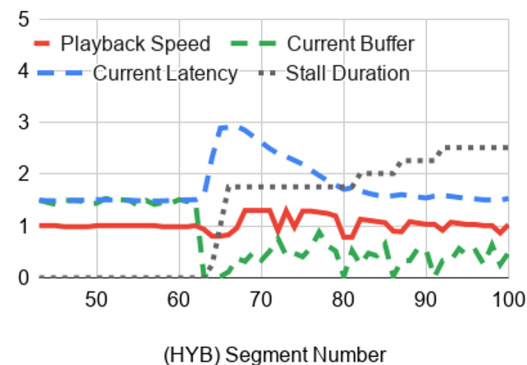
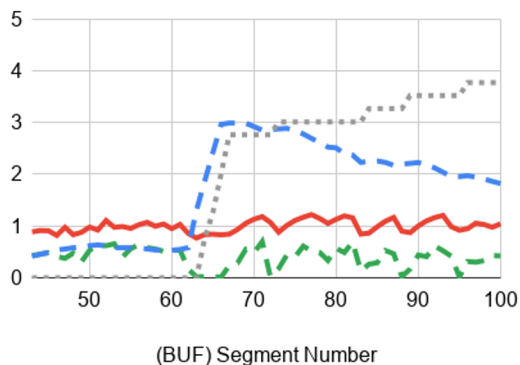
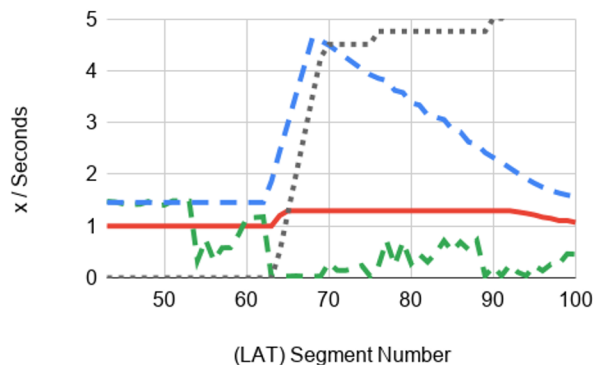
*Adapt the playback speed to achieve the target latency without risking a stall*

		Scenario 1	Scenario 2	Scenario 3
Configurable parameters	Safe buffer	0.5s		
	Target latency	1.5s		
	Playback speeds	0.7 – 1.3x		
	Current buffer	0.3s	0.5s	0.7s
	Current latency	Any value	1.2s	2.0s
	Playback speed	0.8x	0.9x	1.25x

Slow down      Slow down a bit      Speed up

# LoL+: Hybrid Playback Speed Control Module

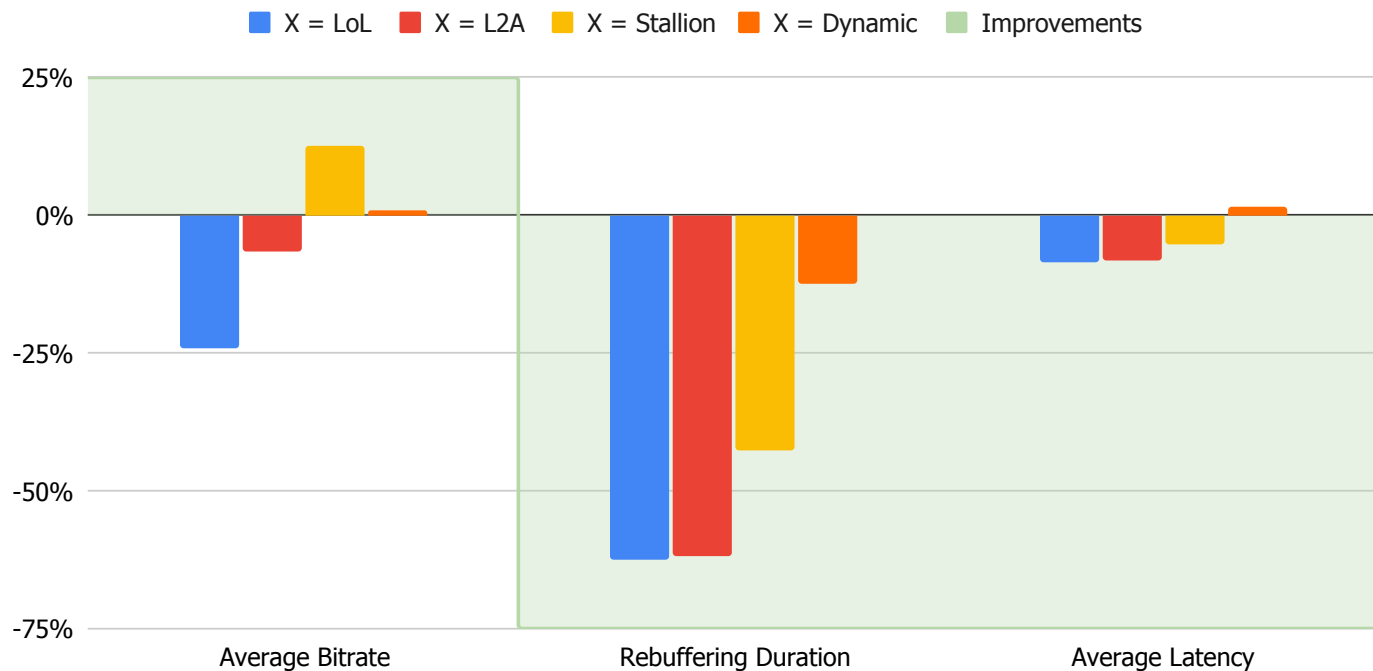
## Sample runs



HYB vs.	Average Bitrate	Rebuffering Duration	Average Latency
LAT	0%	-38.9%	-4.5%
BUF	-3.7%	-60.2%	-1.7%

# LoL+ against LoL and Benchmarks

## Key observations

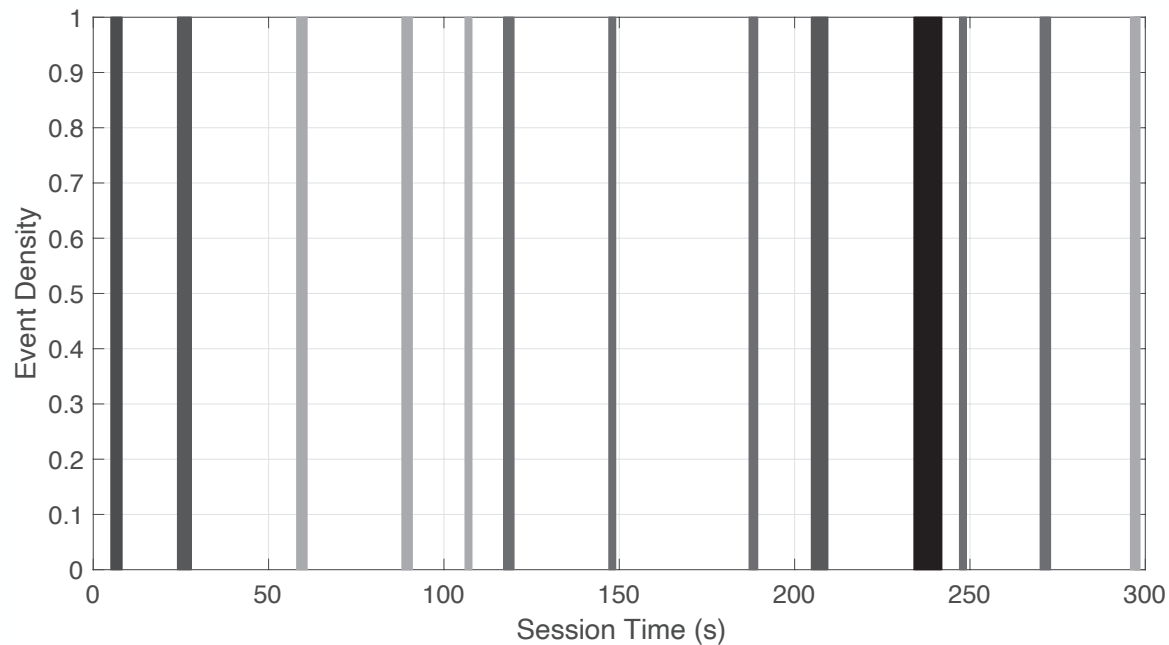


# Content-Aware Playback Speed Control (CAPSC)

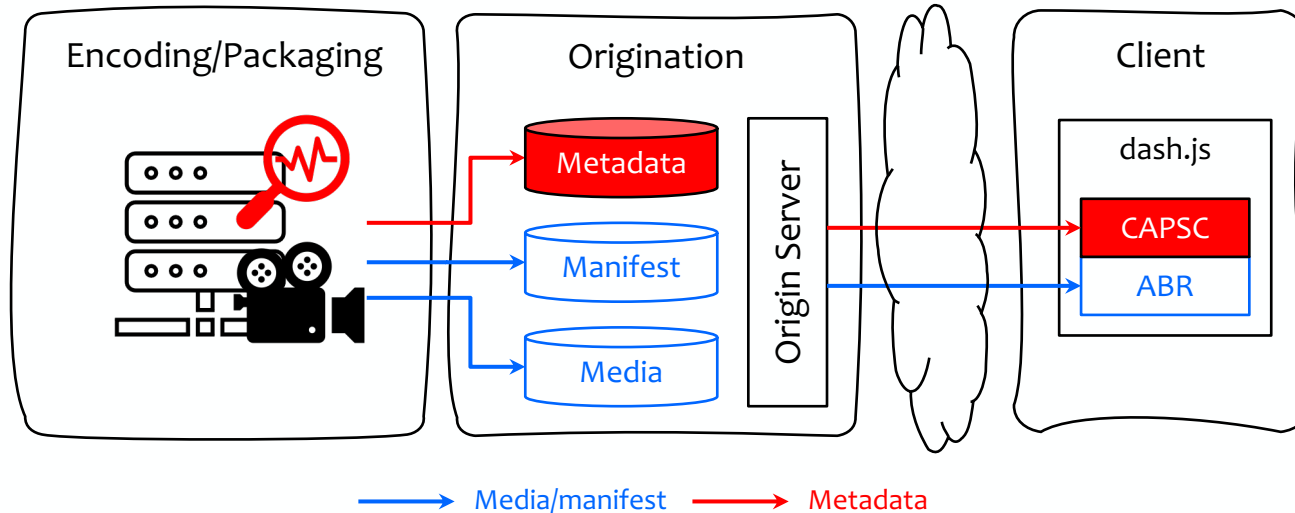
- Playback speed changes may occur at a time when the video and/or audio content is more important or more sensitive to the playback speed
- Viewers are more likely to pay attention to such content and notice the playback speed changes



# Compute a Single Magical Parameter: Event Density



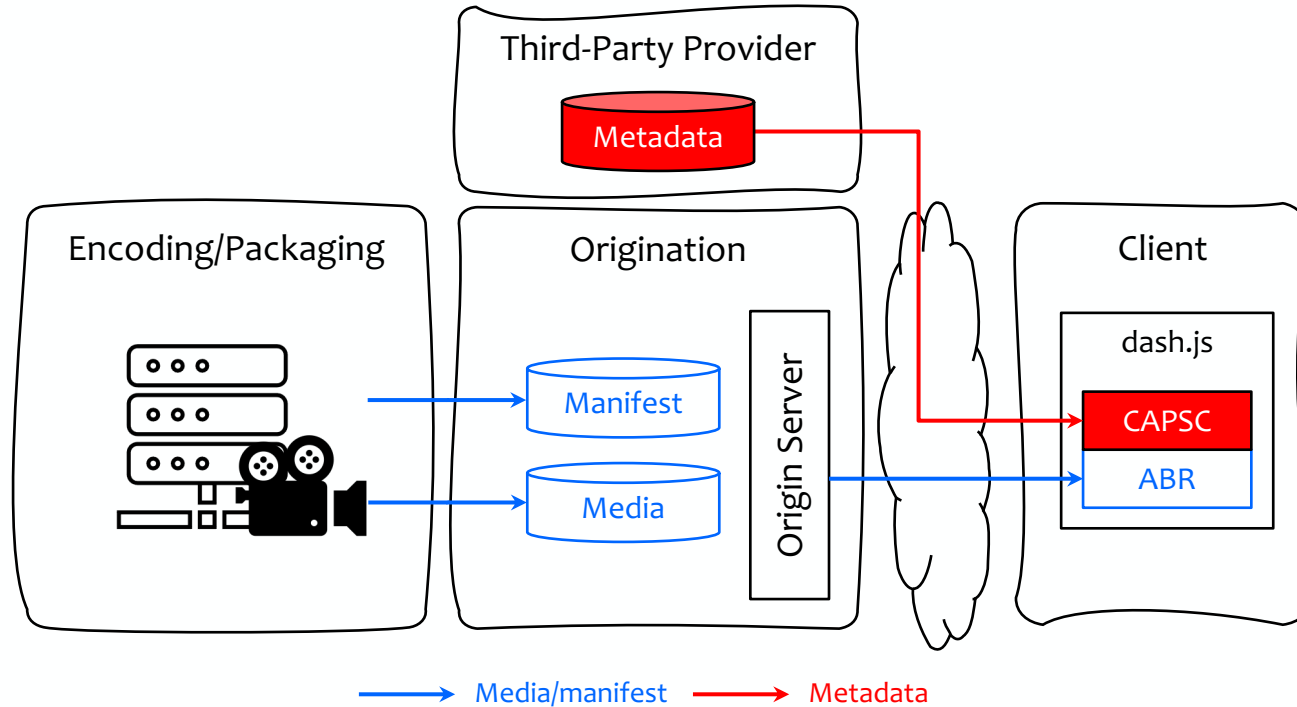
# (In-Band) Signaling from the Encoder



The red-colored parts indicate the new (or modified) elements



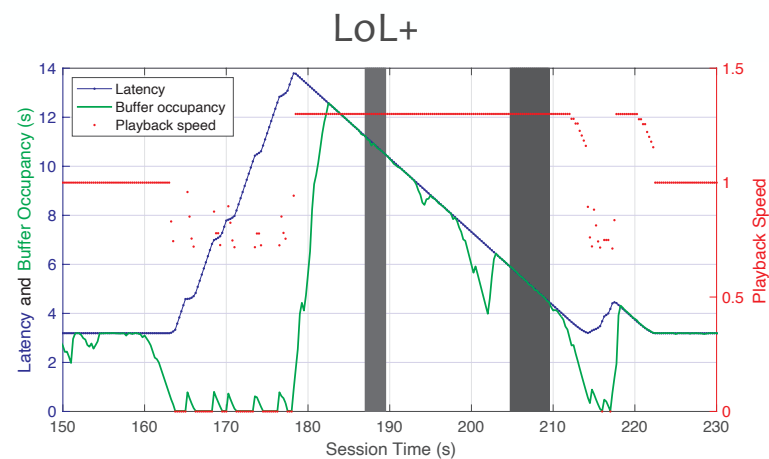
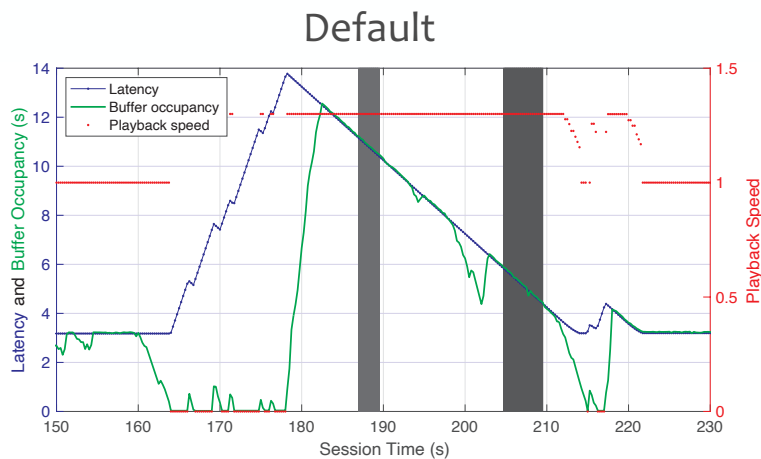
## (Out-of-Band) Signaling from a Third Party



The red-colored parts indicate the new (or modified) elements

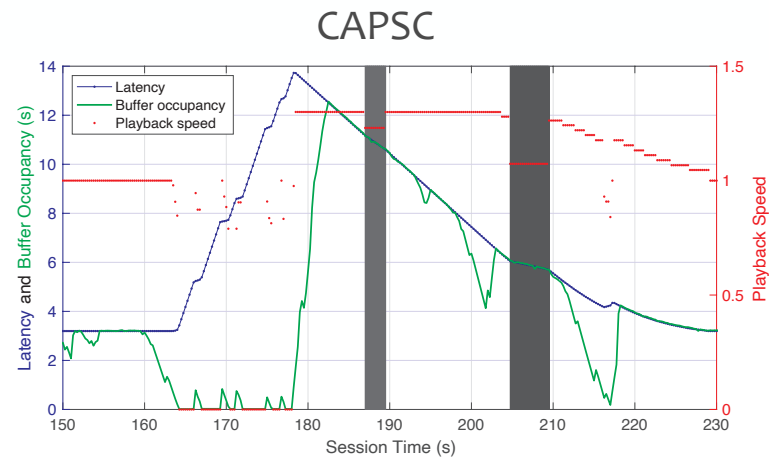
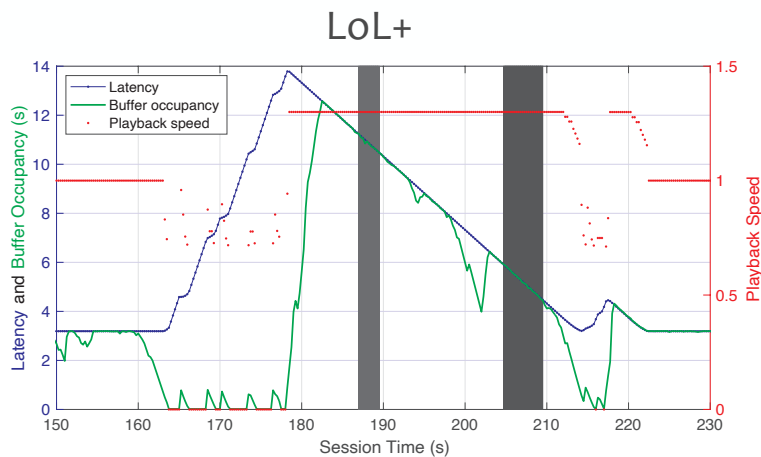
# Default vs. LoL+

$L_{target}$ : 3s,  $B_{low}$ : 1s,  $D_{max}$ : 0.3



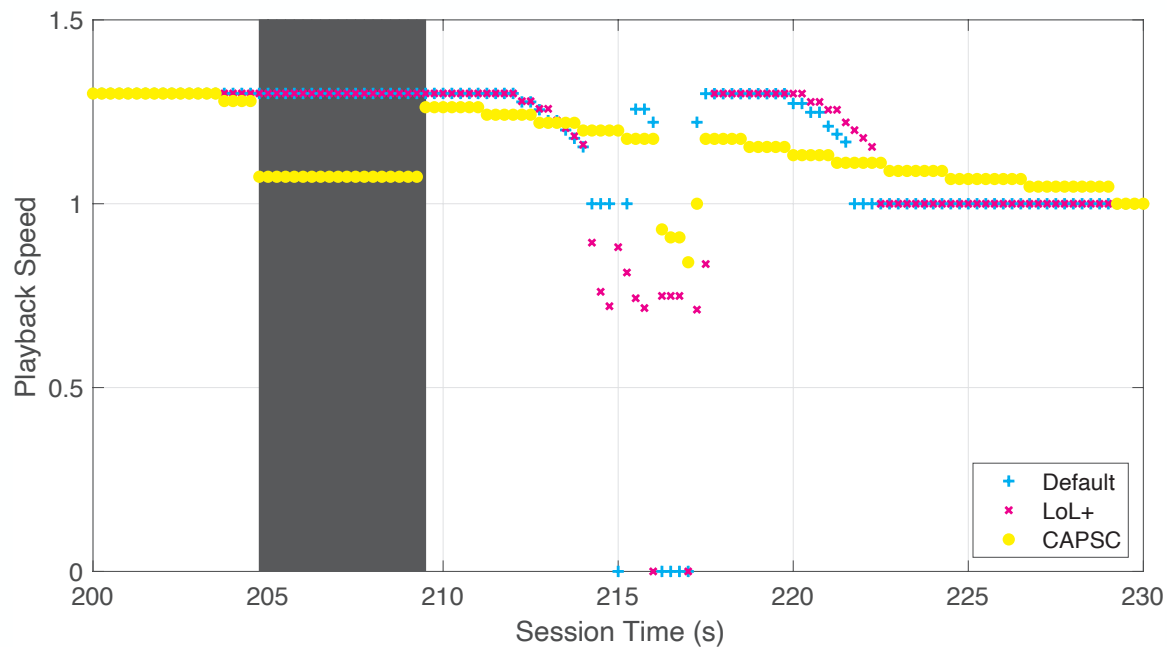
# LoL+ vs. CAPSC

$L_{target}$ : 3s,  $B_{low}$ : 1s,  $D_{max}$ : 0.3

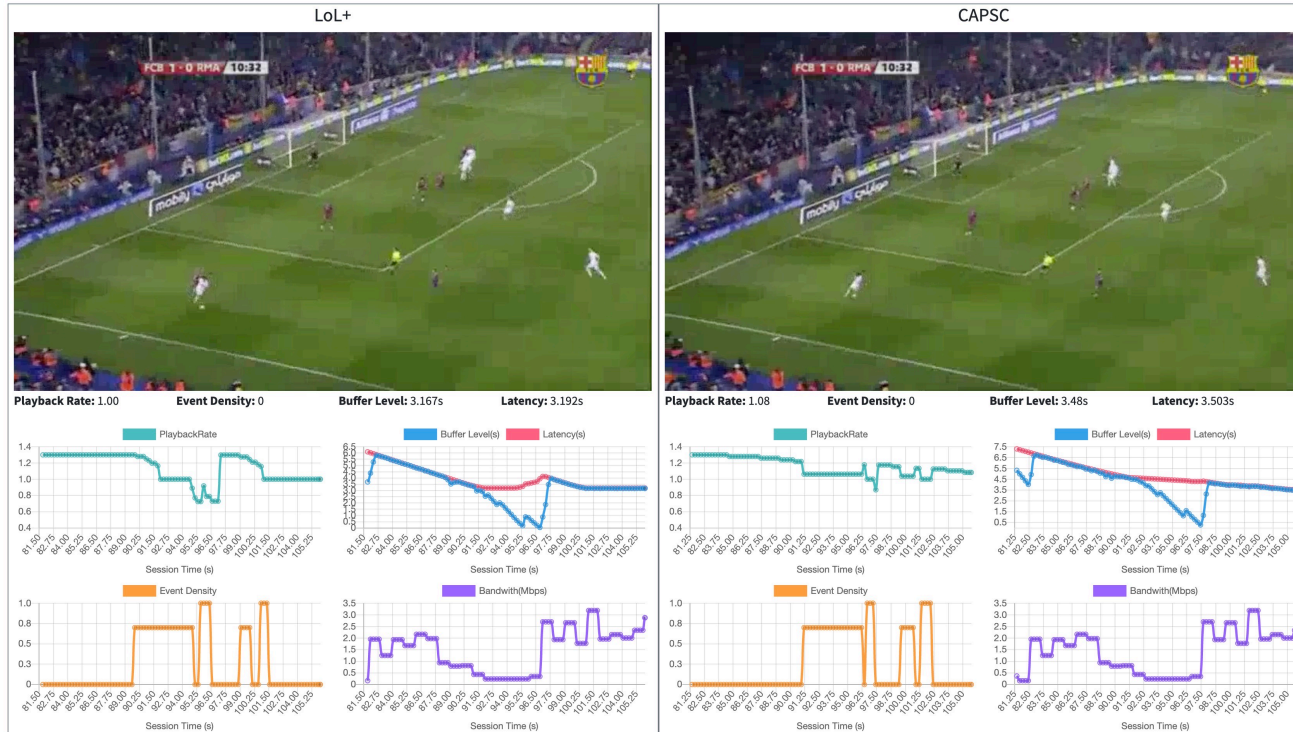


# Zooming in

$L_{target}$ : 3s,  $B_{low}$ : 1s,  $D_{max}$ : 0.3

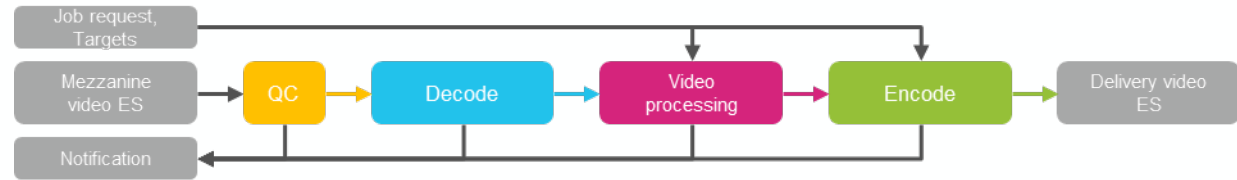


<http://streaming.university/demo/mmsys21-caps>



# Content Preparation for Streaming: Workflows

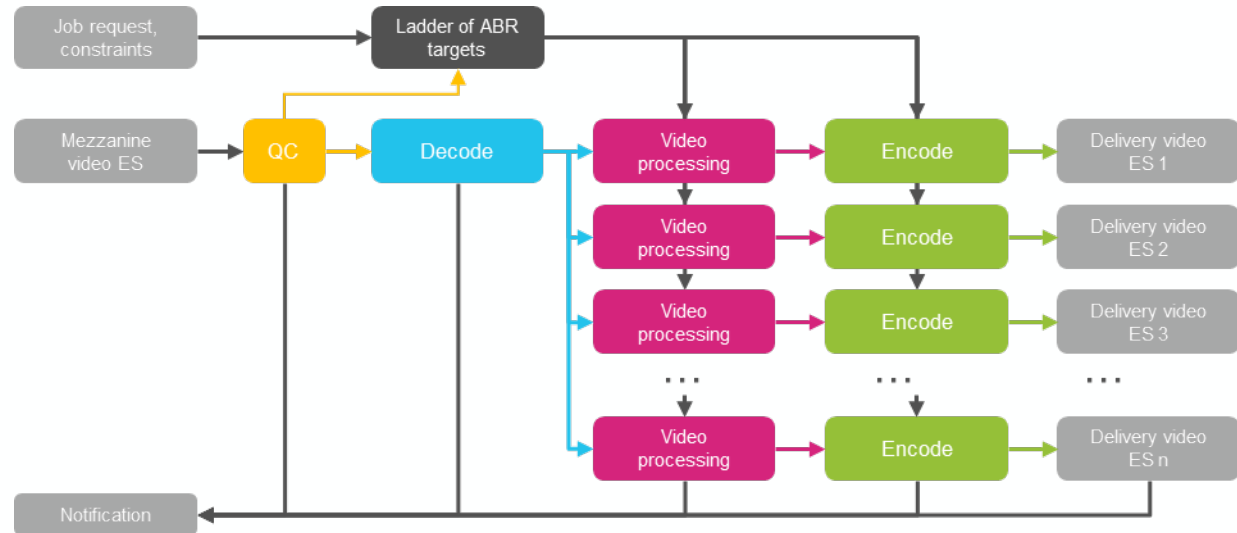
- Single rate transcoding



- ABR transcoding

- Key operations

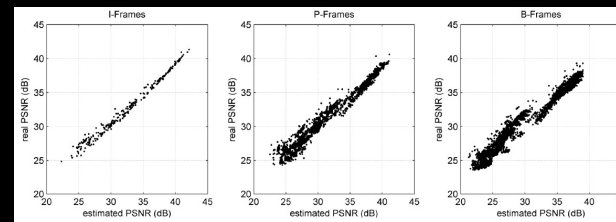
- QC: Quality checks
- Ladder design = CAE
- Pre-processing
- Efficient encoding
- Conformance checks



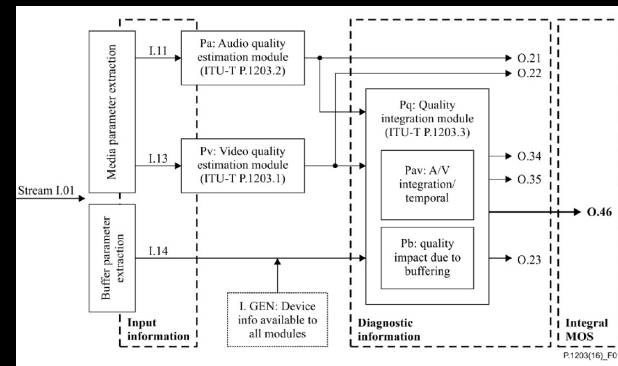
# Input Quality Checks

- Essential for most practical streaming systems
  - If the input is bad, nothing good can be delivered by the system
- Several techniques are available
  - Using QPs and distributions of DCT coefficient
    - Eden 2007, Vanam & Reznik 2019, etc.
  - Parametric metrics
    - ITU-T P.1203.1
      - A recent standard, accounts for many effects such as upscale factor, temporal quality variations, etc.
  - Non-reference metrics
    - BRISQUE (Mittal et al., 2012)
    - BLIINDS (Saad et al., 2010)
    - STAIN (Chu et al., 2012)
    - etc.

Accuracy of PSNR estimation (Eden 2007):

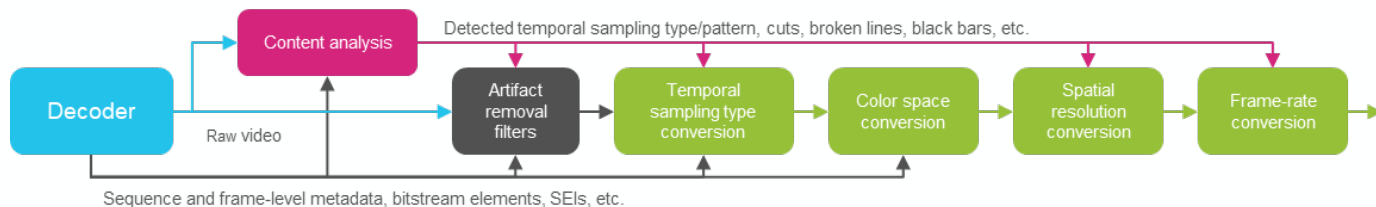


Block diagram of ITU-T P.1203:



# Pre-Processing Chain

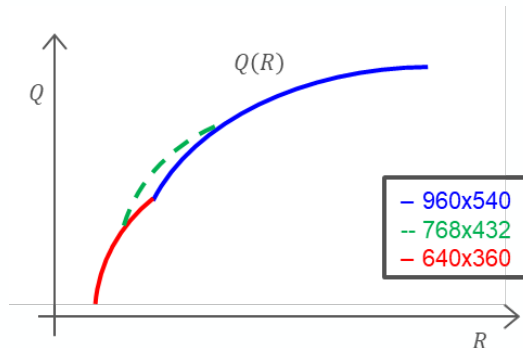
- Key functions
  - **understand** what type of content it is (which may be different from the way it was previously encoded)
    - progressive / interlace / telecine, cadence type, field order, etc.
  - **minimize** artifacts introduced by prior generation encoder or sampling process
    - blocking, ringing, broken lines, temporal noise, etc.
  - **perform conversion** from source format to format needed for delivery, this includes conversions of
    - spatial resolution
    - chroma sampling type (4:4:4, 4:2:2, 4:2:0, different kinds of 4:2:0)
    - frame rate
    - temporal sampling type (progressive, telecine, interlace, field order)
    - color (gamma, matrix, primaries, EOTF, mastering display color volume, display-related metadata, etc.)
- Typical pre-processing chain





# On Choices of Video Resolutions for Streaming

- In theory, the more resolutions are available, the better:
  - Allows better quality/rate tradeoffs:



- In practice, the choices are constrained by
  - content owners
  - industry forum guidelines: DVB, HBBTV, DTV, etc.
  - capabilities of playback devices
- Also important is to consider
  - distributions of native resolutions of players
  - the closer they can be matched, the better

## DVB recommended resolutions

### 10.3 Luminance Resolutions and Frame Rates

A Player that supports HD content shall support the decode and display of pictures with the resolutions in Table 17 and Table 18 at all supported frame rates.

NOTE 1: This does not preclude the use of other resolutions within an Adaptation Set, however, a limited number of resolutions are listed here to ease Player testability.

NOTE 2: The resolutions in the table are the resolutions in the Representations within an Adaptation Set. These may not be the same as the final display resolution, and are thus independent of region specific variations that are prevalent in Broadcast TV.

Table 17: Luminance Resolutions for progressive content

Horizontal @maxwidth	Vertical @maxheight
1 920	1 080
1 600	900
1 280	720
1 024	576
800	450
652	480
768	432
720	404
704	396
640	360
512	288
480	270
384	216
320	180
192	108

Table 18: Luminance Resolutions for interlaced content

Horizontal @maxwidth	Vertical @maxheight
1 920	1 080
704	576
544	576
352	288

A Player that supports UHD TV content shall support the decode and display of pictures with the resolutions shown in Table 19 in addition to the resolutions in Table 17 and Table 18.

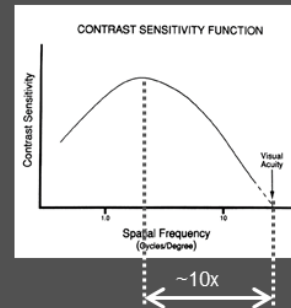
NOTE 3: This does not preclude the use of other resolutions within an Adaptation Set, however, a limited number of resolutions are listed here to ease Player testability.

Table 19: Luminance Resolutions for UHD TV Progressive Content

Horizontal @maxwidth	Vertical @maxheight
3 840	2 160
3 200	1 800
2 560	1 440

For service continuity, reducing the frame rate may be beneficial at lower bitrates, so lower frame rates than are found elsewhere in the present document are needed. A Player shall support frame rates formed by a division by 2 and 4 of those of the frame rate families defined in clause 10.4 that it supports.

## Range of resolutions



Note: 10x downsampled videos start losing details that are most prominently visible in normal reproduction setting.

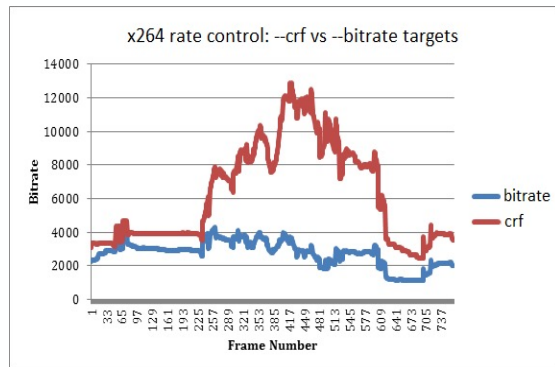
This suggests that the range of resolutions that can be used for encoding should not be larger than 10.

Coincidentally, this is precisely the range supported by the DVB ladder.

# Ensuring Interoperability

- VBR/HRD control
  - for ABR delivery all streams must be capped VBR!
  - the use of highly-variable VBR encoding may confuse clients and cause buffering
  - typically, maximum bitrate cap is set to about 10-35% above average bitrate
  - must be lower than next target bitrate in the ABR encoding ladder
  - decoder buffer size must also be limited
- GOP length and type
  - GOP length must be shorter or equal than GCD of delivery segment lengths
  - E.g., for 4,6, and 10-second segments,  $\text{GOP} \leq \text{GCD}(4,6,10) = 2$  seconds
  - GOP length may also be affected by the need to support SSAI / splicing
  - closed GOP, SAP type 1 is a requirement for HLS
- Profiles and levels
  - H.264 Baseline profile is needed for legacy devices (e.g., mobiles prior to 2012)
  - Main profile is adequate for streams of up to 720p
  - High is better for 1080p and beyond
  - Level must be sufficient to allow given resolution, framerate, bitrate, CPB size
- Reference frames, B frames
  - for legacy devices (e.g., mobiles prior to 2012) – no B frames, 1 reference
  - most STBs can support up to 4 reference frames, 3 B-frames

Example of uncapped VBR behavior:



Example level/profile combinations:

Profile	Level	@codec Parameter (avc1 sample entry)	@codec Parameter (avc3 sample entry)
Constrained Baseline	2.1	avc1.42c015	avc3.42c015
Constrained Baseline	3.0	avc1.42c01e	avc3.42c01e
Main	3.0	avc1.4d401e	avc3.4d401e
Main	3.1	avc1.4d401f	avc3.4d401f
High	3.0	avc1.64001e	avc3.64001e
High	3.1	avc1.64001f	avc3.64001f
High	3.2	avc1.640020	avc3.640020
High	4.0	avc1.640028	avc3.640028

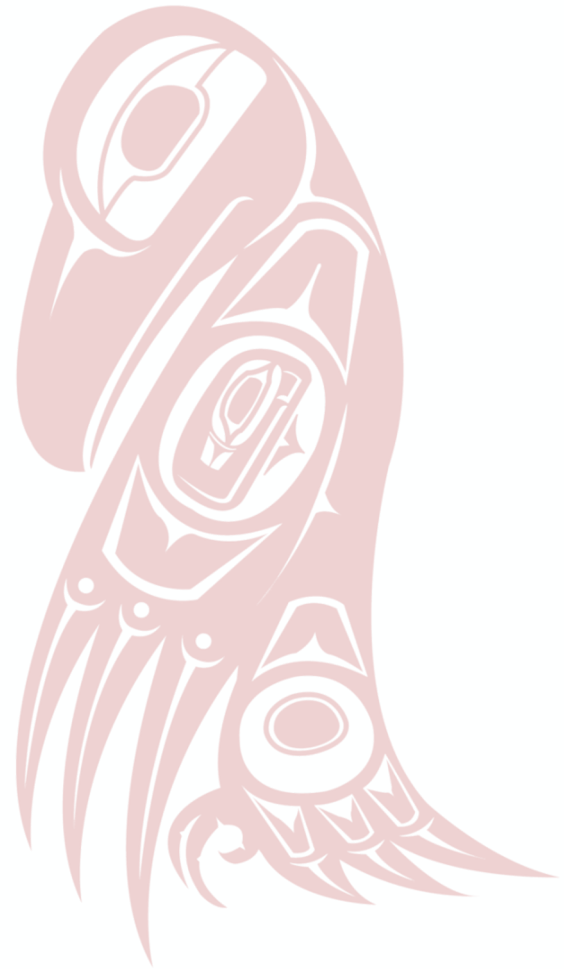
## See the Attached Decks for

- *Per-title, content and context-aware encoding*
- *Optimizations for multi-codec streaming*
- *Optimizations for multi-screen streaming*
- *HTTP streaming and CDN performance*



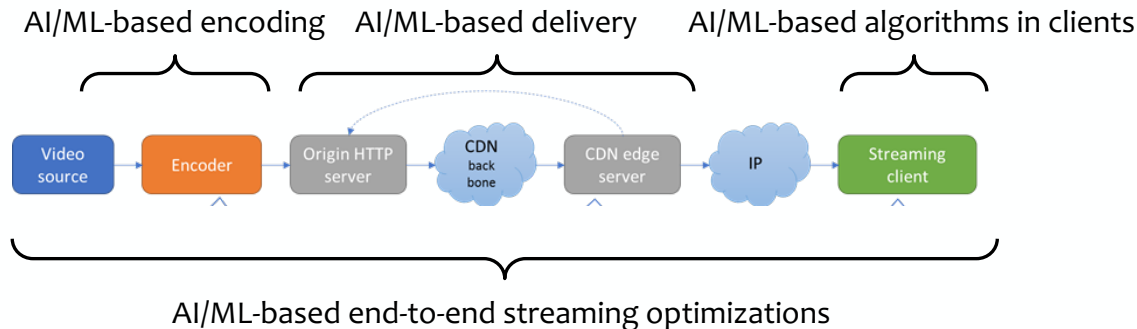
## Ongoing Research and Open Issues

- *AI/ML optimizations for streaming*
- *Quality-aware streaming*



# AI/ML-Based Optimizations for Streaming

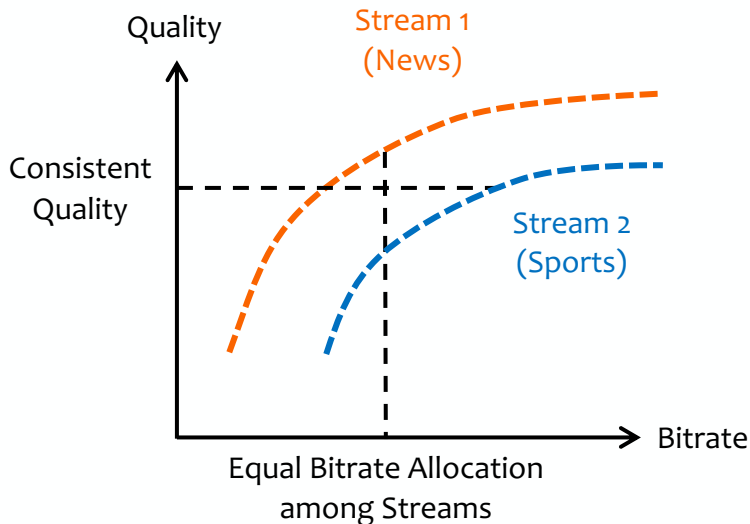
- There are many possibilities and already ongoing efforts:



- Some notable activities:
  - Ongoing work on AI-based audio and video codecs in MPEG/ITU-T JVET, MPAA and AOM
  - Uses of ML in design of modern era perceptual quality metrics
  - Uses of AI/ML for content and context-aware encoding
  - Designs of open streaming statistics databases enabling use of ML techniques
  - Special sessions at many conferences, including ICIP

# Extending the Idea to Optimization across Streams

- Same principle applies to both:
  - In-stream: Temporal bit shifting between segments
  - Across-streams: Bit shifting between streams



## Temporal Pooling

Viewers react differently to glitches for different content types, and they forgive in different time scales

- A young viewer (likely to have longer-term memory) watching sports on a big screen vs.
- An elder viewer (likely to have short-term memory) watching news on a smaller screen

## Spatial Pooling

We want controlled unfairness (which is fairness in quality not bitrate) among viewers

- Example: You are watching football on a 65" screen and your wife is watching a food show on a tablet
- Example: You are watching sports on a phone and two others are watching news on a 48" screen

# Houston, We Have So Many Problems!

- Content formatting
  - Each asset is copied multiple times
    - Different audio/video codecs
    - Different (regional) frame rates
    - Different HDR formats
    - Different container formats, encryption modes
  - Huge cost for encoding/packaging/storage
  - Inefficiencies in CDN caching/distribution
- Across platforms
  - Lack of consistent app behavior
  - Varying video features, APIs and semantics
- Playback
  - Partial profile support
  - Switching glitches
  - Audio discontinuities
  - Ad splicing problems
  - Long-term playback instability
  - Request protocol deficiencies
  - Memory problems, CPU weaknesses
  - Scaling (display) issues
  - Variable HDR support
  - Unknown capabilities, ...
- Security
  - Piracy and restreaming
  - Account sharing, use of VPNs/proxies, ...

## Q&A

- Remember that the slides will be posted at <https://ali.begen.net>
- You can send any questions to [ali.begen@ozyegin.edu.tr](mailto:ali.begen@ozyegin.edu.tr) and/or [yreznik@brightcove.com](mailto:yreznik@brightcove.com)





## Further Reading and Links

- A survey on bitrate adaptation schemes for streaming media over HTTP
  - IEEE Communications Surveys & Tutorials, 21(1):562–585, 2019 (DOI: 10.1109/COMST.2018.2862938)
- DASH-IF guidelines on low latency
  - <https://dashif.org/news/low-latency-dash/>
- dash.js low-latency examples
  - <https://reference.dashif.org/dash.js/latest/samples/> (Click on the Live Low Latency tab)
- Public test assets for client developers and live emulator service
  - <https://testassets.dashif.org/>
  - <https://livesim.dashif.org>
- Extensions for FFmpeg to support LL-DASH
  - <https://ffmpeg.org/ffmpeg-formats.html#dash-2>
- Apple's videos on LL-HLS
  - <https://developer.apple.com/videos/all-videos/?q=low%20latency>

## Further Reading and Links

- MPEG/IETF Workshop on session management and control for MPEG DASH
  - <https://mpeg.chiariglione.org/about/events/workshop-session-management-and-control-mpeg-dash>
- cta-wave/common-media-client-data
  - <https://github.com/cta-wave/common-media-client-data>
- cta-wave/common-media-server-data
  - <https://github.com/cta-wave/common-media-server-data>
- dash.js CMCD reporting
  - <http://reference.dashif.org/dash.js/latest/samples/advanced/cmcd.html>
- Video monitoring dashboards with near real-time edge logs and CMCD KPIs
  - <https://tinyurl.com/akamai-cmcd-dashboard>