LiveVideoStack | 公开课

The Performance of LL-HLS and LL-DASH Streaming Players

Bo Zhang（张博）
Staff video research engineer,
Brightcove inc.

About myself

张博现供职于美国波士顿的Brightcove公司，从事视频技术的研发工作。他的主要研究方向包括 video content delivery, low-latency and real-time streaming, video playback, IP networking, 并代表Brightcove公司参与CMAF, DASH等视频标准的制定工作。他也是多个 视频标准委员会的成员，包括ISO/IEC SC29 working groups (MPEG), INCITS L3.1, DASH Industry Forum, CTA-WAVE。他本科毕业于华中科技大学，硕士毕业于美国University of Cincinnati，博士毕业于美国George Mason University。他曾在video streaming及wireless communications领域发表多篇论文，其中一篇曾 获得ACM MSWiM 2011年最佳论文。

Introduction

现有基于HTTP的Low-latency live streaming protocols

- LL-DASH (DVB, DASH Industry Forum, MPEG)
- Community-driven LHLS（Twitch, Twitter's Periscope)
- Apple's LL-HLS (Apple)
- LL-CMAF (MPEG)

Server到client的延迟可控制在3秒以内（美东到美西），甚至低于2秒（同城）

应用场景: 体育直播，在线教育，在线博彩，和各种需要即时互动的应用。

Low-latency live streaming的设计思路 - 服务器端

服务器端：编码器将一个数秒长的video segment切割成多个数百毫秒的video chunks。在第一个chunk生成之时，马上发送给客户端，而无需等待整个segment的产生。

CMAF (Common Media Application Format)或者Fragmented MP4可作为LL-DASH和LL-HLS的共享封装格式，以达到“一流两用”的目的。

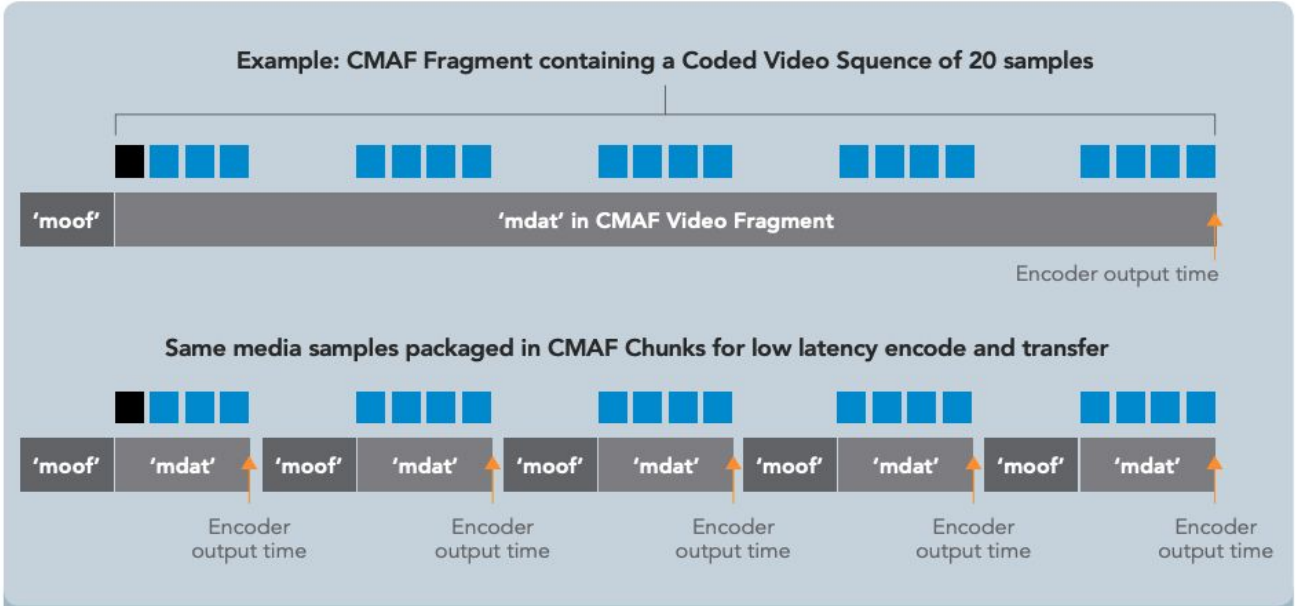
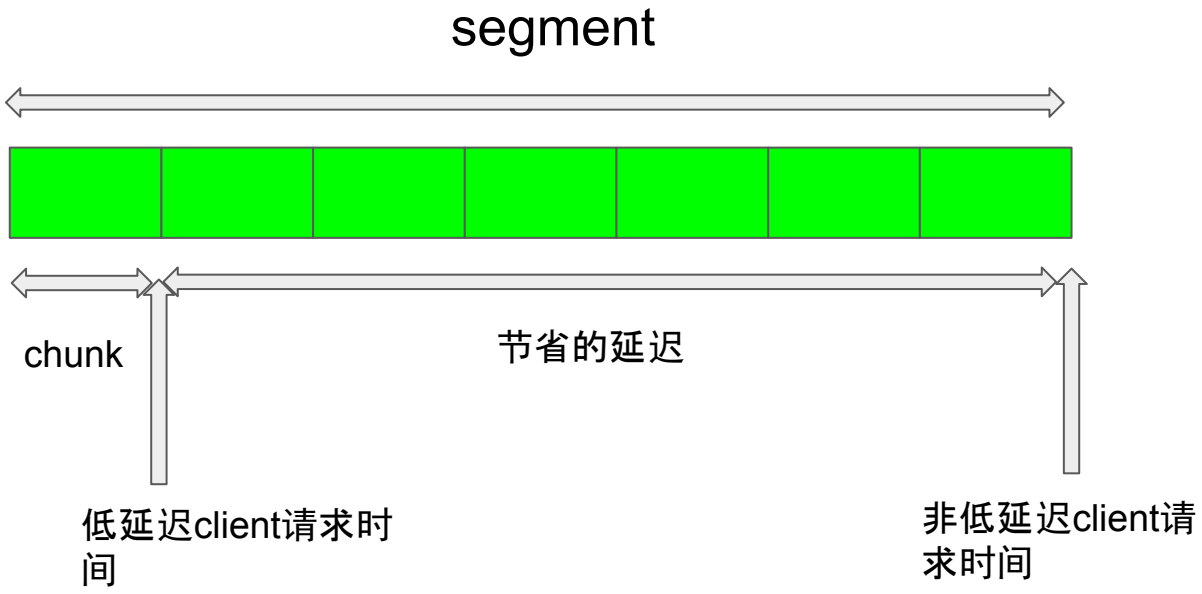


Figure 2: Chunked encoding of a CMAF segment

Low-latency live streaming的设计思路 - 客户端

客户端：在每个video segment的第一个chunk生成当时，客户端及时向服务器请求该segment，而不做任何等待。客户端采用Fetch API（而非XHR API)下载每个chunk。下载完成后不做任何等待，将chunk交给浏览器的Media Source Extension (MSE)。



LL-HLS and LL-DASH server/client support

客户端：

- LL-HLS: Apple's native AVPlayer (iOS, macOS, iPadOS), Android ExoPlayer, Hls.js, Shaka player, Theo player.
- LL-DASH: Dash.js, Shaka player, Theo player, Android ExoPlayer

编码器及CDN：

- LL-HLS: Apple's HLS reference tools,Wowza streaming engine, Ateme Titan live, …
- LL-DASH: FFmpeg, GPAC, Akamai, ...

Objective of this talk

对市面上的多款LL-HLS and LL-DASH players做性能评测，包括Shaka player, Hls.js, Apple AVPlayer, 以及Dash.js及其两款针对latency的改进版。

性能评测基于以下几种指标，

- latency
- average stream bitrate
- bitrate switch的频率
- rebuffering的频率
- Bytes downloaded
- 播放速度的稳定性

测试结果以论文的形式已发表在ACM MMSys 2021。

我个人也利用业余时间为Brightcove Video.js player开发了LL-DASH的功能，https://github.com/maxutility2011/videojs_lowLatencyDash。

Experiment setup

- Low-latency测试流的搭建
  - LL-HLS测试流: 使用FFmpeg和Apple HLS reference tools生成
  - LL-DASH测试流：使用FFmpeg及node-gpac-dash搭建
  - 支持multiple bitrates
  - Video segments使用fragmented MP4格式封装：每个segment时长4秒，每个chunk时长1秒
  - 测试内容：big buck bunny (动画）

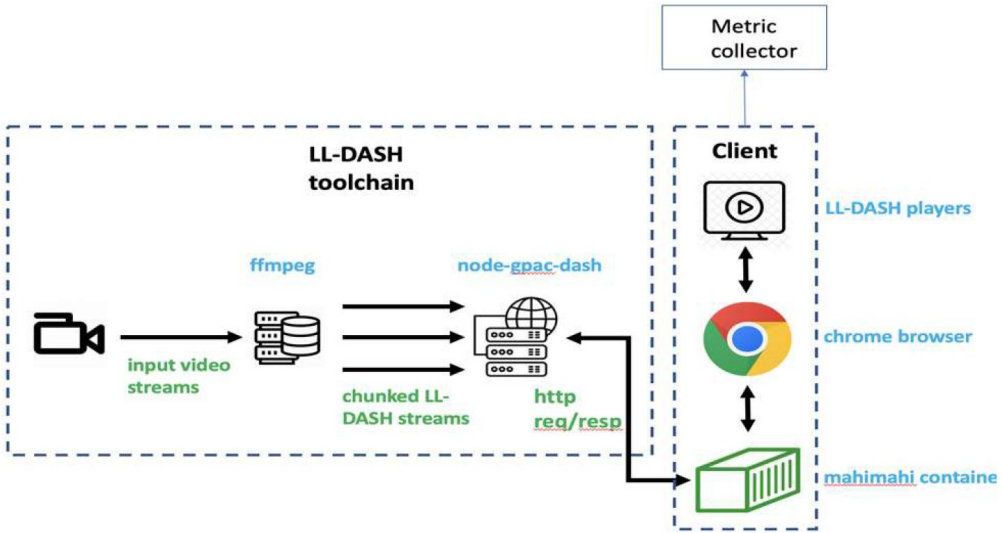
	Bitrate 1	Bitrate 2	Bitrate 3
Bitrate (kbps)	279	925	1253
对应的video resolution (pixels)	320 x 180	640 x 340	768 x 432

Experiment setup - LL-DASH

使用FFmpeg作为LL-DASH repackager,

- 注入端(ingest)可使用任意低延迟协议，比如 RTP, SRT，WebRTC。
- FFmpeg输出的LL-DASH流（包括chunked segments及MPD文件）可使用HTTP/1.1 Chunked Transfer Encoding (CTE) 推送到origin server（云存储, 或本地存储）。

使用node-gpac-dash从本地存储中逐一读取一个segment中的每个chunk, 并使用HTTP/1.1 CTE将chunks推送给clients.

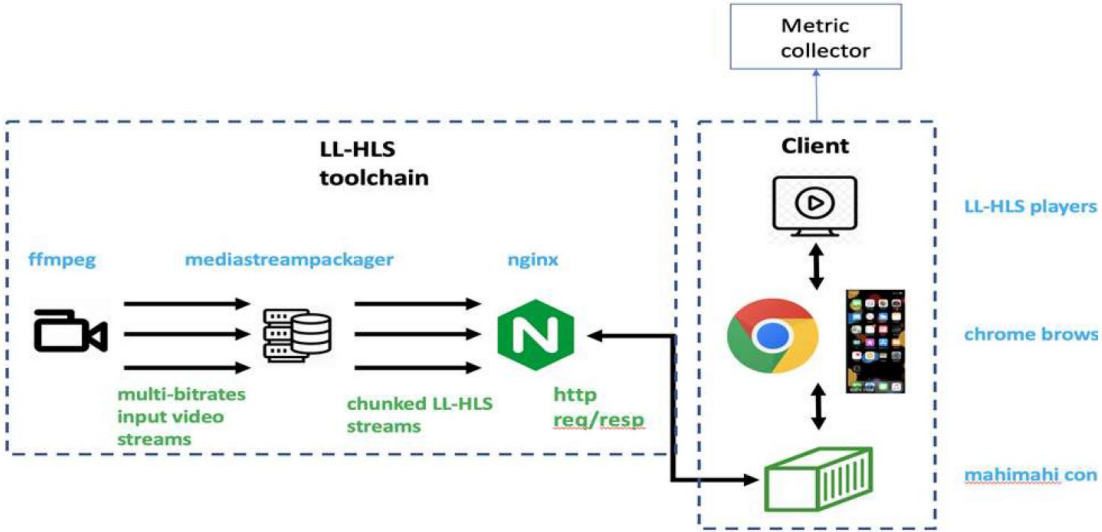


Experiment setup - LL-HLS

使用FFmpeg产生multi-bitrates streams，并推送给Apple's mediastreampackager。

mediastreampackager将input streams切割成chunk并推送给origin server/cdn，并且产生m3u8文件。

使用Nginx作为origin server。在Nginx内运行Apple提供的LL-HLS server PHP script来处理客户端请求。



Network emulation

- 我们设计了一种基于真 实网络数据，高精度，可重演的 network emulation框架

  - 框架基于Mahimahi network emulator [3]。

  - 使用美国两家大型无 线网络运营商Verizon和T-Mobile的真实4G网络数据 (network trace) 来模拟真实网络环境。

- Mahimahi network emulator是一款基于容器的模 拟器。

Mahimahi容器仅有的一个虚 拟网络接口可以根据 给定的network trace来发送或接收数据包，以 实现精准的网络环境模拟。

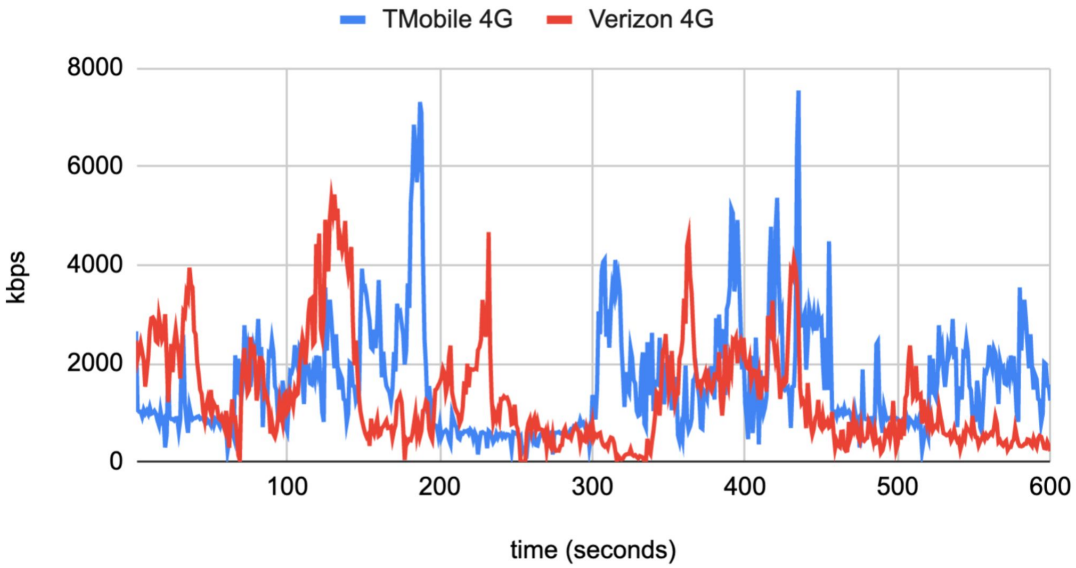
在Mahimahi容器中可以运行任意程序，比如基于 浏览器的players，或者native players。其中运行的程序只能通过虚拟接口从服务器接收video segment，其下载网速因此受到network trace的限制。

通过Mahimahi network traces的重演 (replay)，我们可以让不同video players运行于完全相同的网 络环境中，从而达到 测试的公平性。

Network emulation

Verizon 4G and T-Mobile 4G网络数据

Available network bandwidth



Bandwidth statistics

Network	T-Mobile	Verizon
Average (kbps)	1607.43	1323.97
Variation (kbps)	1147.60	1075.80
Min. (kbps)	148.5	1.178
Max. (kbps)	7545	5433

基于T-Mobile 4G网络模拟的测试结果

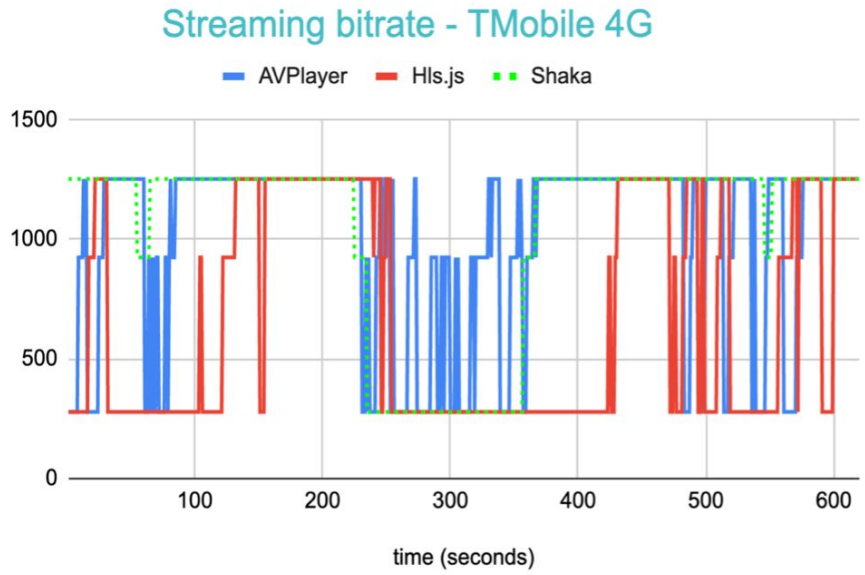
测试对象

LL-HLS	LL-DASH
Hls.js on Google Chrome	Dash.js on Google Chrome
Shaka player on Google Chrome	Dash.js + L2ALL on Google Chrome
AVPlayer on iOS 14	Dash.js + LoL on Google Chrome

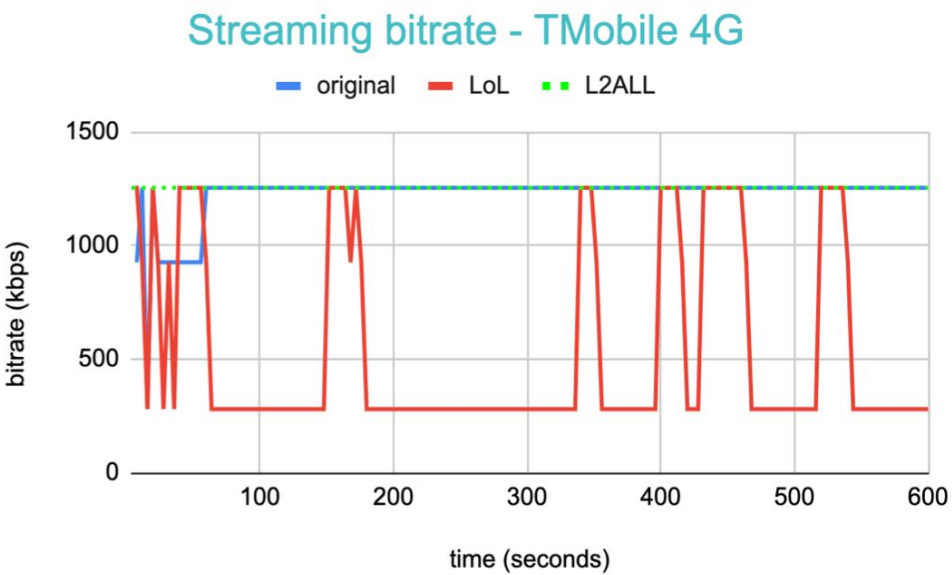
L2ALL和LoL是两种针对低延迟播放器的bitrate adaptation算法，两者都采用了机器学习的算法。

Stream bitrate

LL-HLS

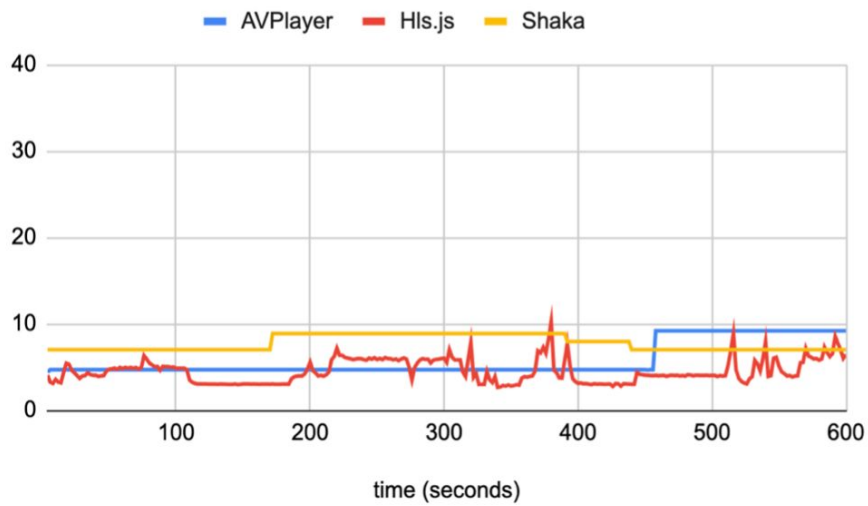


LL-DASH

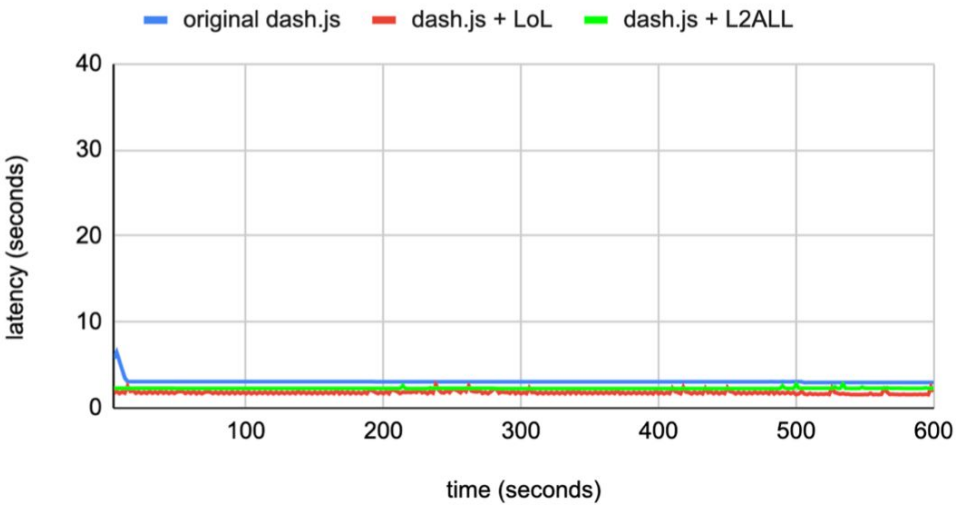


Streaming latency

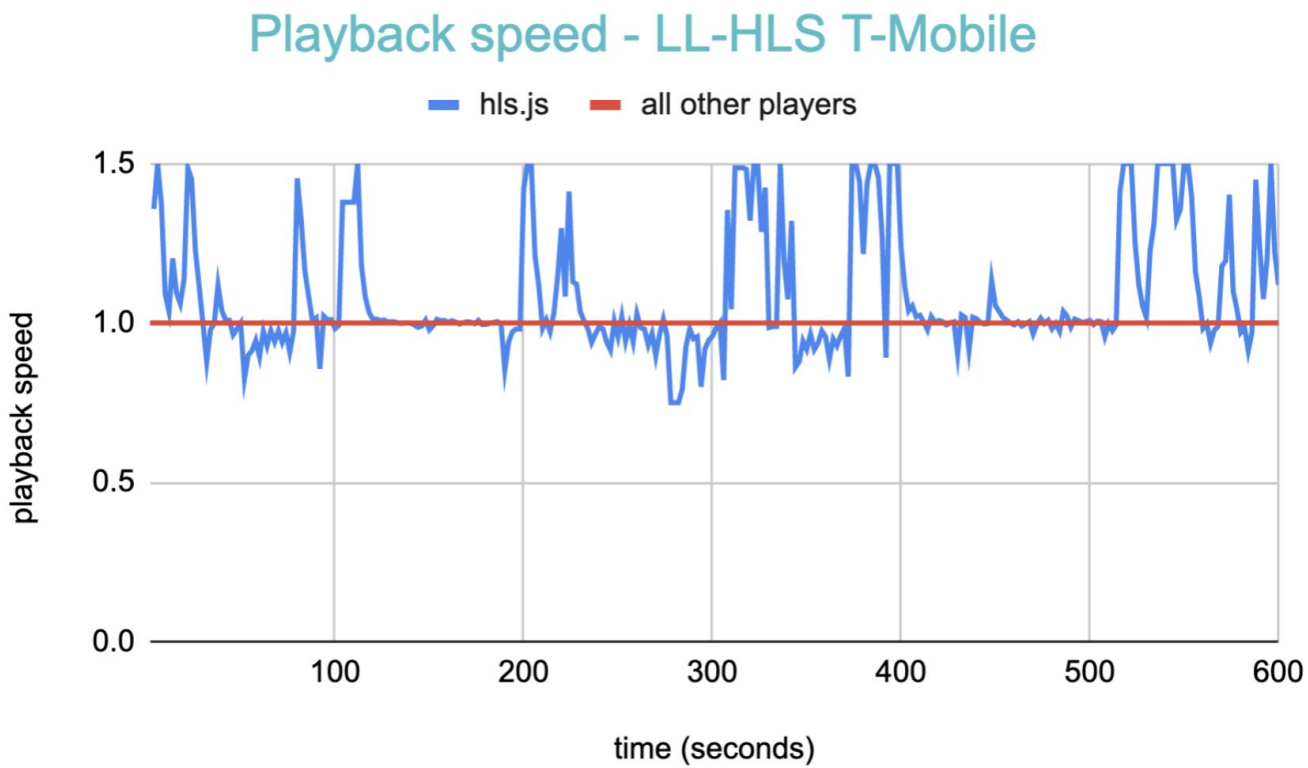
LL-HLS



LL-DASH



Playback speed variation



Statistics

Player	Average bitrate (kbps)	Average height (pixels)	Average latency (secs)	# of media objects downloaded	Megabytes downloaded	# of rebuffering events	# of rendition switches	Playback speed variation
Hls.js	783	311	4.48	965	155.54	43	50	3.62
Shaka	1043	378	7.78	621	81.23	18	8	0
AVPlayer	1037	378	5.82	703	91.63	1	72	0
Dash.js	1225	426	3.06	151	92.57	1	5	0.23
LoL	53	248	1.78	152	42	69	28	1.62
L2ALL	1251	432	2.28	151	94.49	13	1	0.42

测评结论

1. 低延迟通常会导致更多的重新缓冲和播放停顿，特别是在不稳定的移动网络中。有些播放器会部分牺牲低延迟，采取更多的缓冲来减少停顿。

2. 除了降低延迟之外，播放器还需要保正延迟的稳定性，避免播放器因为延迟变化过大而不断调整播放速度，从而造成用户体验的降低。

3. LL-HLS在m3u8文件中列出每个segment中的每个chunk，让播放器逐一请求每个chunk，造成服务器端需要处理大量的请求，从而影响可扩展性。

LL-DASH采用HTTP chunked transfer encoding让服务器主动推送新产生的chunk给播放器，而播放器只需请求每个segment，而不是每个chunk。因此，服务器端承受的负载会成倍地少于LL-HLS。

测评结论

4. 如果视频流提供多个bitrates，我们发现有些低延迟播放器会避免选择下载high bitrate stream。原因可能是担心下载high bitrate stream会导致下载网速跟不上stream bitrate而导致的重新缓冲。

5. 不同播放器的bitrate adaptation算法会选择偏向不同的性能指标，比如，有些播放器会优先考虑最小化延迟，有些则选择适当牺牲延迟，增加缓冲，以保证播放流畅度，有些也会选择同时兼顾低延迟和bitrate。

6. Low-latency streaming在视频行业正受到极大的关注。LL-HLS和LL-DASH两种技术相互竞争，业内对于选择那种技术，尚未形成共识。很多公司认为LL-DASH在技术上更成熟更合理，对于Apple在LL-HLS的设计提出一些质疑，但是奈何Apple在音视频领域影响巨大，而（不得不）选择优先支持LL-HLS。LL-DASH在技术上也并非完美，而且标准制定需要业内各家公司协商完成。

LiveVideoStack | 公开课

Q & A