基于延迟目标的 DASH.js 播放器分析

Piers O'Hanlon BBC R&D 伦敦，英国
piers.ohanlon@bbc.co.uk

Adil Aslam BBC R&D 伦敦，英国
adil.aslam@bbc.co.uk

arXiv:2304.13551v1 [cs.MM] 26 Apr 2023

摘要

我们针对一系列延迟目标和配置选项，分析了 dash.js Dynamic 自适应 HTTP 流媒体（DASH）播放器中三种自适应比特率（ABR）算法的低延迟性能。我们在 DASH 测试平台上进行了实验，该平台支持使用一系列源自真实世界的网络配置文件进行测试。我们的实验有助于更好地理解延迟目标如何影响体验质量（QoE），以及不同算法对其目标的遵循程度。我们发现，在 dash.js v4.5.0 中，默认的 Dynamic 算法能够实现最佳的总体体验质量。我们表明，尽管其他算法能够在更低延迟下实现更高的视频质量，但代价是增加卡顿。我们分析了 L2A-LL 在测试中的较差性能，并进行了修改，结果显示其性能得到显著改善。我们还强调，某些低延迟配置设置可能会损害性能。

CCS 概念

• **信息系统** → **多媒体流媒体**；• **网络→网络仿真**。

关键词

DASH、低延迟、流媒体、性能评估

1 引言

对于通过传统广播和 IP 流媒体分发的直播内容，理想情况下，IP 服务至少应达到与广播相当的延迟。有多种方法可以实现低延迟 IP 流媒体，但在不同延迟目标下的性能尚未得到充分研究。了解传输延迟与体验质量之间的适当权衡非常重要。

本文分析了广泛使用的基于 Web 的 DASH 播放器 dash.js [6]，的低延迟性能，考察了其三个自适应码率算法下的一系列延迟目标和配置选项。默认的自适应码率算法称为 Dynamic[14]，，是一种混合算法：播放缓冲区较短时使用基于测量吞吐量的吞吐量算法，其他情况下则使用其基于缓冲区占用率的李雅普诺夫算法（BOLA）[15]。

Learn2Adapt 低延迟 (L2A-LL) [7] 算法使用凸优化，根据前一个片段对延迟的影响，预测未来片段表示的最佳选项。低延迟 (LoL+) [2, 9] 算法提供了一种体验质量最大化决策方案，同时利用基于自组织映射的启发式预测模型和学习模型。每个分段边界都被用作机会，根据内部定义的加权体验质量模型，从给定的表示中推导出预测体验质量最高的选项。

通过在我们的 DASH 测试平台上开展一系列实验，我们阐明了运行低延迟流媒体时延迟与体验质量之间可能存在的权衡。我们的测试平台支持在基于浏览器的播放器与低延迟流媒体服务器之间进行基于轨迹的网络仿真和指标采集，并使用分块传输模式传输时长为 3.84 秒的片段。这些指标使我们能够分析延迟、视频质量和卡顿等因素对性能的影响。我们采用 ITU-T P.1203 体验质量模型及其实现 [12]，生成总体体验质量评分。我们还考察了其他配置选项的影响，例如 FastSwitching 算法 [14]，该算法控制播放器是否会尝试获取替换用的更高质量分段。

尽管其他分析侧重于尽可能降低延迟，但我们发现，在较低延迟（~3s）下，体验质量显著低于较高延迟下的体验质量，不过在约 8s 时开始趋于平稳。我们的贡献如下：

- 基于延迟目标比较 dash.js 的三种低延迟算法。
- 发现，在 dash.js v4.5.0 中，默认的 Dynamic 算法实现了最佳的整体体验质量，并且对延迟目标的遵循程度最高。
- 表明默认启用的快速切换算法会浪费带宽，并且通常不会改善整体体验质量。
- 阐明 L2A-LL 算法表现不佳的原因，并证明我们所提出改进的益处。
- 证明不同 Dash.js 版本之间存在显著的性能差异。

其他研究主要探讨了在固定低延迟下单个低延迟自适应码率算法的性能，往往侧重于实现尽可能低的延迟，而未考虑在更高延迟下的性能 [7, 9]。此外，比较 dash.js 三种低延迟自适应码率算法的研究并不多。最近的一项研究[17] 进行了比较，但未考察不同的延迟目标。

在许多情况下，与其追求可实现的最低延迟，不如让不同传输机制、平台和设备之间保持一致的延迟。这对于可能在体育比赛中听到他人提前为某个事件欢呼的观众而言非常重要。这还有助于确保每个人在社交媒体上的反应时间大致一致。由于延迟、可靠性和质量相互关联，追求最低延迟势必会降低可靠性或质量，或两者兼而有之，因此，务必考虑具体情境实际需要何种延迟，不要超出必要范围。

© 2023 BBC。本文为作者版本。仅供您个人非商业用途。不得再分发。最终版本发表于第*14*届 *ACM* 多媒体系统会议论文集（*MMSys '23*），https://doi.org/10.1145/3587819.3590971。

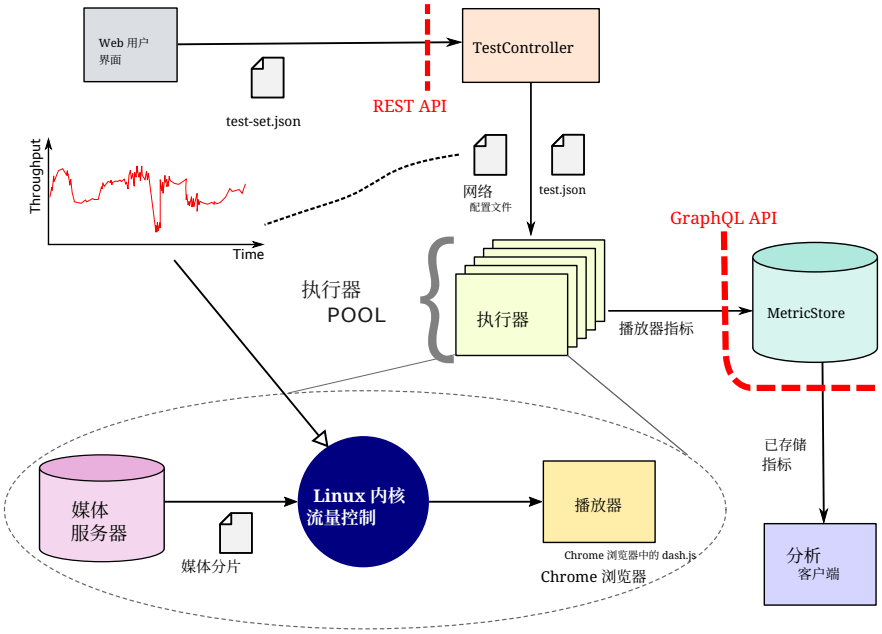


图 1：DASH 测试平台架构

本文其余部分的结构如下：首先，我们在第 2 节介绍实验测试平台和方法的详细信息；接着，在第 3 节介绍我们的结果；然后，在第 4 节提供进一步分析和建议。我们在第 5 节总结相关工作，最后在第 6 节总结主要发现，并提出未来研究的潜在方向。

2 实验

我们使用自己的 DASH 测试平台 [1] 评估了 dash.js 的性能。我们选择开发自己的测试平台，而不是使用其他测试平台，以便完全控制其各个方面的行为。我们使用四种不同的网络配置文件进行了一系列实验，模拟各种网络条件，从而能够比较三种自适应码率算法的性能。

2.1 DASH 测试平台

图 1 展示了 DASH 测试平台的整体架构。测试平台的核心是 TestController，负责将测试集加入队列并运行测试集。基于 Web 的用户界面允许通过 JSON 文件向 TestController 提交测试集，该文件描述了要运行的每组测试。此 JSON 文件还可用于控制测试平台的配置。首先，它允许指定测试中使用的流：是直播流还是点播流、包含何种音频/视频/字幕组件，以及流的时长。其次，它可以配置测试中使用的 DASH 播放器软件，以及该播放器应如何配置。目前，测试平台支持基于 dash.js JavaScript DASH 实现和 GStreamer 媒体工具包的播放器。在我们的案例中，我们使用的是运行 dash.js 的 Google Chrome 浏览器。第三，可以设置该测试要运行的会话数。最后，还可以指定在每个测试会话运行期间应应用的网络配置。

当 TestController 在测试集中运行测试时，它会将该测试的各个会话分配给执行器池，每个执行器一次负责运行一个会话。每个执行器都是一台独立的物理 Linux 机器，因此所有测试彼此完全独立。我们使用 Linux 内核内置的流量控制功能，根据该测试所选的特定网络配置文件限制其网络接口的数据传输速率，并确保这些接口使用通常为 1500 字节的最大传输单元（MTU）。执行器会配置并启动测试描述中指定的播放器，该播放器将实时播放测试流，并将有关其性能的指标报告回 MetricStore。

指标既来自内置的 dash.js DVB DASH 指标报告功能，也来自自定义 JavaScript 函数。通常，指标数据类型可分为两组：基于事件的指标和周期性的指标。基于事件的指标通常会在事件发生时报告，并附带额外的上下文信息，例如：HTTP 事务、重新缓冲时段和表示切换。因此，每次运行中基于事件的指标数量可能有所不同，在某些情况下也可能为零。周期性报告用于监控某些指标，例如缓冲区级别和延迟。周期性报告会按照固定间隔提供参数值，间隔通常为 500 毫秒，由测试平台设定。每次运行中，这些指标通常具有相同数量的条目。所有指标均带有时间戳。

清单中的可用视频表示见表1，这些表示代表了业界所使用的表示。分段采用通用媒体应用格式（CMAF），长度为3.84秒，由四个0.96秒的块组成。内容由一段新闻视频素材构成，该素材已编码为H.264 恒定比特率（CBR）流，并

分辨率	192x108	256x144	384x216	512x288	704x396	896x504	704x396	960x540	1280x720
比特率（Kbps）	86	156	281	437	827	1,374	1,570	2,811	5,468
帧率	25	25	25	25	50	25	50	50	50

表1：我们实验所使用的视频码率阶梯

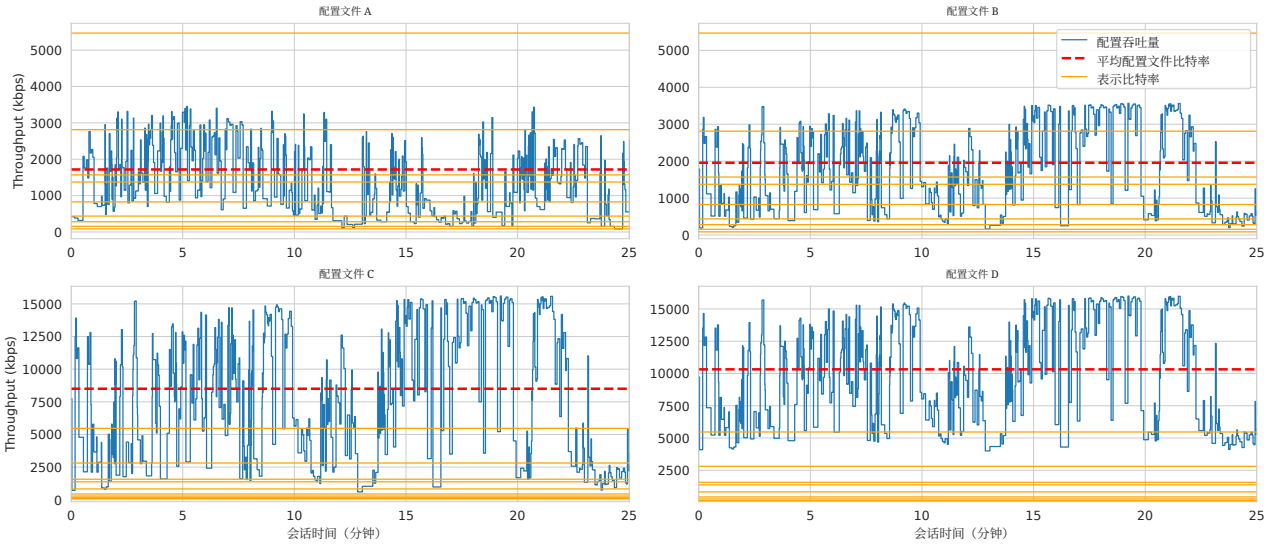


图2：网络配置文件（含视频码率阶梯）

重新打包，以便作为实时低延迟编码媒体用于Testbed 媒体服务器。测试流还包含一个128kbps的音频表示。我们采用的体验质量（QoE）指标与内容无关，因此所使用的内容类型不会影响结果。

图2中的网络配置文件用于测试运行，其数据源自从 BBC iPlayer 流媒体会话中采集的数据。橙色线条表示经过测试的表示的比特率，展示了这些比特率与各种网络配置文件中的吞吐量之间的对比。

2.2 方法

以典型地面广播（DVB-T）的延迟作为实时 IP 流媒体的主要比较基准，并据此选择待测试的延迟目标。这种比较对于大规模普及十分重要，因为用户/消费者会期待产品能与现有媒介相媲美。

选择了多个延迟目标，使其覆盖 DVB-T 现有的延迟范围；该范围通常为 6 - 8 秒。选定的数值为 3、5.5、8 和 15 秒。前三个数值间隔相等，以便在整个范围内设置测量点。选择 15 秒作为目标，也是为了将其用作相对稳定的运行点，并与传统（非低延迟）实时流媒体进行比较。

使用的 dash.js 版本为 v4.5.0，于 2022 年 9 月发布。dash.js 的配置参数大多保持默认值。部分参数经过修改，以实现特定功能、利用现有的播放选项，或改变测试场景。我们将 最大漂移 设置为 5 秒，将 播放速率 设置为 0.17。播放速率经过设置，使得在最大速率下，流在 30 秒的时间窗口内大约能够追赶上 5 秒的延迟。

对于使用 Dynamic 和 L2A-LL 自适应码率算法的测试用例，追赶机制采用默认选项。LoL+ 算法使用其专属的自定义追赶机制，因此该机制仅用于 LoL 的测试用例。

对于吞吐量估算技术，使用了默认的“moof”解析方法，而不是较旧的数据块方法。选择滑动窗口作为移动平均技术，这同样是默认方法。

为确保每次运行所提供的结果不受先前测试影响，我们将 *lastBitrateCachingInfo* 和 *lastMediaSet- tingsCachingInfo* 均设置为 false。在正常使用中，启用这些参数可以提升会话的启动性能，因为它们允许将此前获取的吞吐量估算结果和表示选择作为决定初始表示请求的因素。

除配置比较测试外，我们在所有测试中均禁用了快速切换功能，因为研究发现该功能会对性能产生不利影响（见第 4.2 节）。

测试场景由特定的延迟目标、三个自适应码率算法之一以及四个网络配置文件之一共同定义。测试平台配置的往返时间（RTT）为 50 毫秒。每个测试场景将运行 1497.6 秒（24 分 57.6 秒）。这一特定时长源于所使用的分段数量

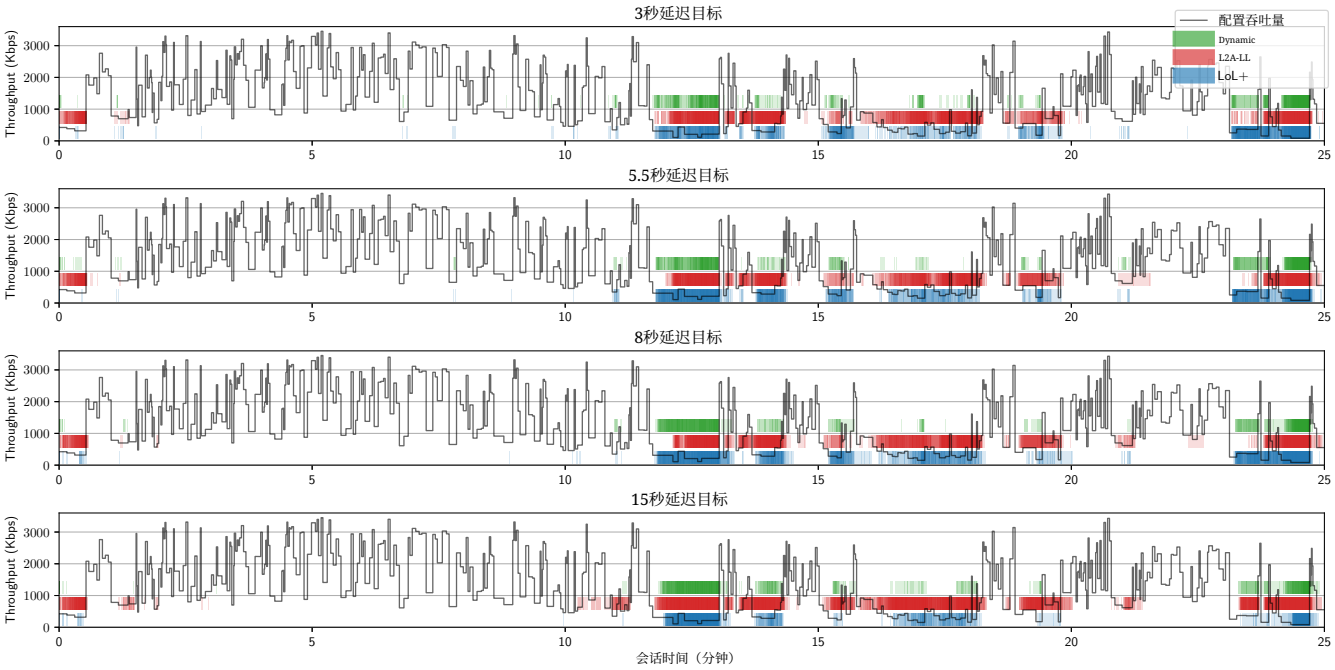


图 3：卡顿热力图（配置文件 A）

，即 390 个，假设以 1 倍播放速率连续播放时，每个片段的时长为 3.84 秒。

每个测试场景均运行了二十次。这是为了确保计算出的任何平均值都具有足够的置信度。

3 结果

本节使用 dash.js v4.5.0 对我们的核心实验进行分析。首先，我们根据所考察的关键指标对其进行拆解，然后再讨论总体体验质量。

3.1 重新缓冲

重新缓冲对体验质量的负面影响在视频点播（VOD）中已得到广泛认可[4, 5, 8]。在低延迟环境下，重新缓冲所耗费时间对体验质量的负面影响大于视频点播场景[5]，因为它会增加播放延迟。

三种自适应比特率算法所经历的卡顿量存在显著差异，卡顿次数和随后重新缓冲的持续时间均是如此。

图 3 展示了卡顿热力图，其中使用一组彩色条带表示每种自适应比特率算法发生的卡顿，并将其叠加在网络配置文件上。在这方面，Dynamic 算法在三者中表现最佳，卡顿次数最少，整体重新缓冲时间也最短。LoL+ 紧随其后，这两项指标均有所增加；L2A-LL 的表现则明显落后于另外两种算法。

提高目标延迟确实会降低卡顿程度，因为更大的缓冲区为算法提供了更充裕的时间进行适应。这些变化反映在体验质量得分的提升上，详见第 3.4 节。

虽然篇幅有限，无法展示其他场景，但我们在不同网络配置文件中观察到了类似的趋势。

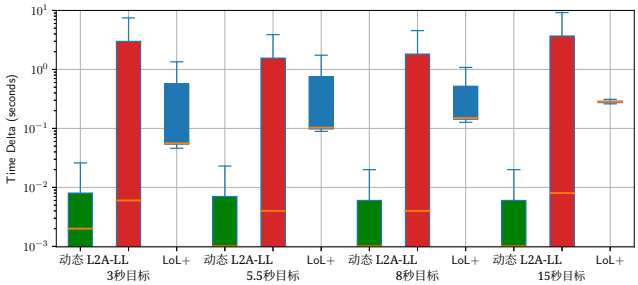
我们预计，减少重新缓冲时间能够使客户端更接近直播延迟目标，下一步将评估这一指标。

3.2 延迟

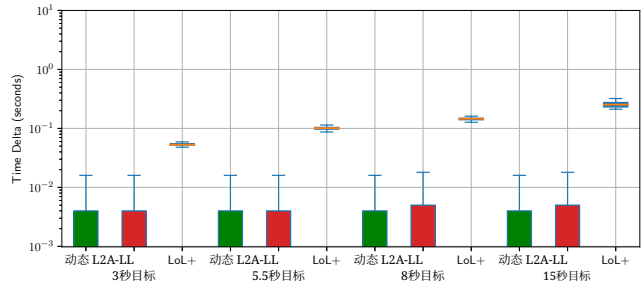
本节将探讨在所测试的延迟目标范围内的性能表现，并分析不同算法对其目标的遵循程度。

此处使用的延迟指标，是指内容在源站可用到客户端将其渲染出来之间的延迟。在低延迟流媒体传输中，内容提供商会选择直播延迟目标，通常会在媒体呈现描述（MPD）中进行配置。客户端应保持在目标值或接近目标值，以免观众在屏幕上观看事件时，被社交媒体上的相关消息或邻居对同一事件早些时候的反应抢先影响。直播延迟值每隔 500 毫秒从 dash.js 客户端获取一次。

图 4 使用箱线图展示了不同算法遵循其延迟目标的情况，其中中位数以橙色线条标示，须线延伸至 1.5 倍四分位距。我们使用对数刻度；在图 4(b) 中，未显示中位数表示其值为零。我们观察到，在两种场景下，Dynamic 与目标的偏差都最小。图 4(a) 展示了最具挑战性的场景下的性能；即使在挑战性最低的配置文件（图 4(b)）中，这一趋势总体上仍然相似，不过置信区间有所收窄，时间差也有所减小。我们注意到，LoL+ 的差值似乎存在小幅系统性增长。

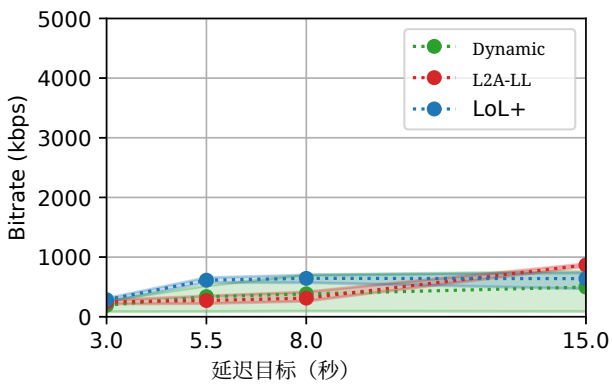


（a）配置文件
A

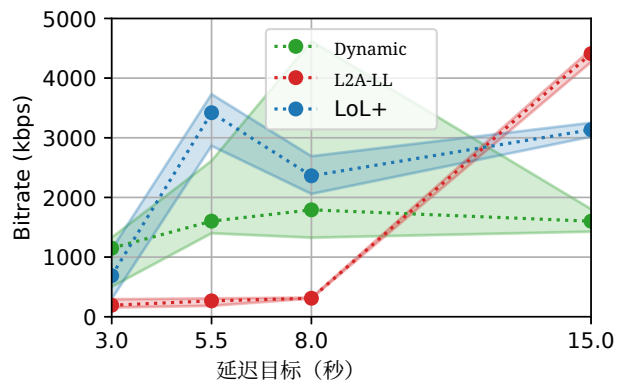


（b）配置文件
D

图4：相对于目标的延迟偏差



（a）配置文件
A



（b）配置文件
D

图5：平均视频比特率

3.3 视频质量

在预测用户对视频点播 [5]的满意度时，所选视频比特率被认为具有相当大的影响。我们首先评估这一指标，随后给出更多结果，表明在直播流场景中，单独使用比特率可能会产生误导，或无法全面反映情况。

如图5所示，我们绘制了20次运行的均值，阴影区域由第5和第95百分位数界定。可以看出，不同算法之间存在显著差异。在最具挑战性的场景（图5(a)）中，所有算法都难以维持高于1000Kbps的速率；而在挑战性最低的情况下（图5(b)），三种算法的比特率通常会在不同延迟目标之间显著波动，其中 Dynamic 提供了最稳定但较低的比特率，而 L2A-LL 和 LoL+ 则达到了更高的比特率。

我们使用公式1计算平均视频比特率（B）。我们会从表示下载次数中移除所有重复请求。

$$\overline{B} = \frac{segment\_duration}{total\_playback\_time} \sum_{R=108p25}^{720p50} (N_R \times B_R) \tag{1}$$

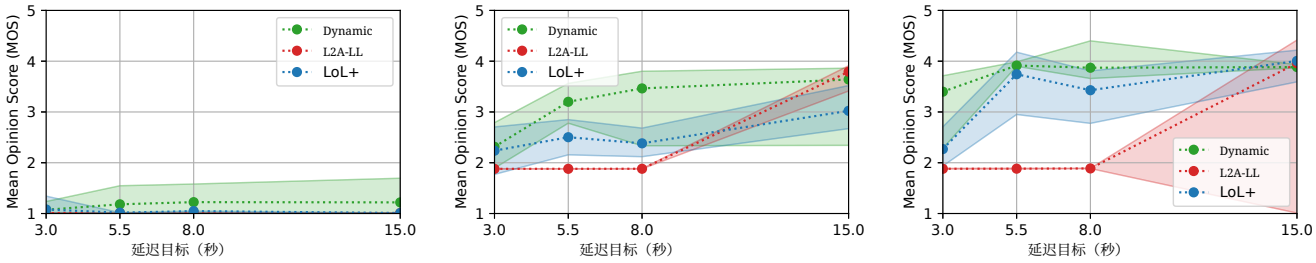
$$Where\ R = Representation$$

$$N_R = Number\ of\ times\ R\ downloaded$$

$$B_R = Bitrate\ of\ Representation$$

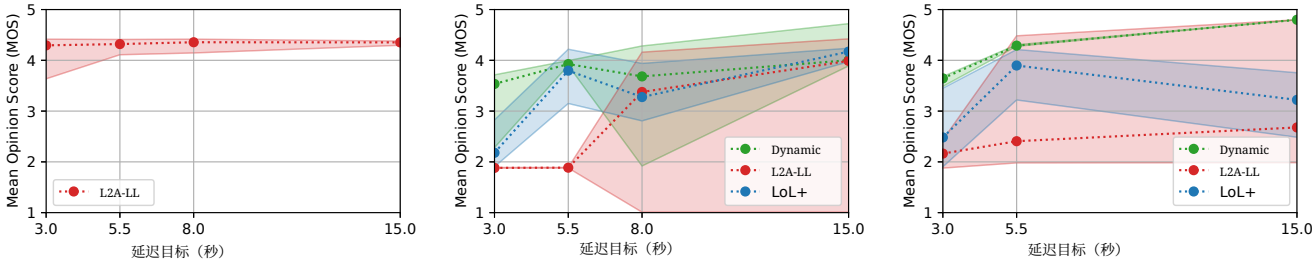
3.4 体验质量

我们采用了 ITU-T Rec. P.1203，一种更全面的总体体验质量定量测量方法，可用于通过参数化方法和基于机器学习的码流分析，对经由可靠传输的自适应视听流媒体服务进行质量评估。我们使用参考实现 [12] 生成总体体验质量评分，形式为平均意见分（MOS）；该评分采用模式0，基于流元数据。我们采用这一体验质量测量方法，因为它经过了大量训练，并通过对一千多个包含HAS典型效应（如卡顿、表示切换）的视听序列进行主观测试验证。该方法不受绝对延迟影响，不过会通过卡顿测量反映延迟变化。相比之下，低延迟特定的



(a) 配置文件 A　(b) 配置文件 C　(c) 配置文件 D

图 6：体验质量 - P.1203 总体测量值



(a) 修改版 L2A-LL　(b) 已启用快速切换　(c) dash.js v4.3.0

图 7：体验质量 - P.1203 总体测量值：进一步分析（配置文件 D）

其他研究中使用的体验质量测量方法 [2, 16] 缺乏主观用户测试依据，而且还包含一个延迟因子，会根据其较低的绝对延迟给予奖励，而不是根据算法对其延迟目标的遵循程度进行评价。

在图 6 中，我们绘制了 P.1023 体验质量总体结果，其中各点表示 20 次运行的均值，阴影区域由第 5 百分位数和第 95 百分位数界定。虽然在配置文件 A 下，所有结果都相当低，但从图 6(a) 可以看到，Dynamic 位居首位；而对于配置文件 C（图 6(b)）和 D（图 6(c)），Dynamic 算法在所有延迟目标下的表现都优于其他算法，直到它们在最高的 15秒目标处全部趋于一致。

4 进一步分析

在本节中，我们探讨结果中的一些特征，讨论这些特征产生的原因，并在适当情况下提出对 dash.js 的改进建议。

4.1 L2A-LL 分析和改进

我们深入考察了 L2A-LL 在低于 15秒目标时（图 6(c)）表现如此不佳的原因，发现这是多种因素共同作用的结果。由于我们的测试采用了 960 毫秒的分块时长，与较短的块相比，dash.js 会花更多时间等待分段变为可用状态；而在低延迟模式下，调度器的超时时间为 300 毫秒，因此在新分段可用之前，自适应码率算法可能会被多次调用。这似乎不会影响其他算法，但对于 L2A-LL，这可能导致算法状态在短时间内被针对每个分段意外地多次更新。L2A-LL 还存在一个特有问题，使情况雪上加霜：它会将最近一个分段的吞吐量估计作为算法输入，却未排除根据初始化分段得出的测量值。初始化分段的问题在于，由于其体积较小（~900字节），且难以准确测量其传输时间，吞吐量估计通常非常低——事实上，dash.js 的吞吐量历史记录中并不包含这些分段。对于低于 15秒目标的流，这些问题共同导致 L2A-LL 陷入较低状态：当它首次尝试切换到较高比特率的表示时，首先会下载一个初始化片段。随后，dash.js 尝试获取下一个媒体分段时，该分段尚未可用；前述问题则会导致 L2A-LL 算法再次运行，这次使用的是初始化片段较低的吞吐量估计。这会促使算法再次切换到较低比特率，从而导致体验质量不佳。15秒目标的情况则有所不同，因为它超过了配置参数 稳定缓冲时间 的默认值 12 秒；该参数限制了 dash.js 在尚未达到最高画质时构建的缓冲区大小。在这种情况下，请求的是前沿之后、已完全可用的分段；而在 L2A-LL 算法运行之前，初始化分段的任何吞吐量估计都会被媒体分段的代表性测量值取代。

我们对 L2A-LL 算法进行了修改，以解决我们所强调的问题：仅当吞吐量估计来自算法之前未使用过且不是初始化片段的片段时，才更新算法状态

	3秒延迟	目标		5.5秒延迟	目标		8秒延迟	目标		15秒延迟	目标	
	Dynamic	L2A-LL	LoL+	Dynamic	L2A-LL	LoL+	Dynamic	L2A-LL	LoL+	Dynamic	L2A-LL	LoL+
平均唯一片段数	391.4	124.5	391.2	392.0	392.0	392.0	392.6	385.6	392.4	393.9	393.6	393.6
平均总片段数	391.4	126.6	391.2	392.0	392.0	392.0	397.3	484.7	436.0	394.7	425.8	424.0
重新请求百分比	0.0%	1.63%	0.0%	0.0%	0.0%	0.0%	1.1%	20.4%	10.0%	0.2%	7.5%	7.1%

表 2：在配置文件 D 上启用快速切换后的片段请求

在其他情况下，它会返回之前计算出的表示质量切换决策。如图 7(a) 所示，修改后的 L2A-LL 实现能够在所有延迟目标下显著提升性能。这些修改将回馈至 dash.js 项目。

4.2 配置选项

我们研究了其他配置选项的影响，包括 FastSwitching 算法 [14] ，该算法控制播放器是否会尝试获取替换用的更高质量分段。

为了了解此功能的行为，我们检查了分段请求，并比较了唯一分段数量与预期分段总数（390 个，见第 2.2 节）的比率，以及播放会话中未发生卡顿的时间百分比。由于这些测试针对的是直播流，且有指定的时间限制（1497.2 秒），如果卡顿时间足够长，就不会请求所有分段；而在某些情况下，由于性能良好且播放速度提高，请求的分段数量可能会略多。我们利用 DASH 指标报告中的 HTTP 事务列表，确定每次运行请求的分段数量以及下载的唯一分段数量。

自 dash.js v4.0 起，快速切换功能默认启用。启用该功能后，预计算法会在缓冲区大于阈值且网络吞吐量足够时，尝试切换到更高比特率表示，并在之前下载的分段播放前将其替换。预计这只会影响会话中的少数分段，且主要出现在延迟目标较高的场景中，因为较大的缓冲区会增加下载到播放的时间。

启用快速切换后，缓冲区阈值级别被设置为 1.5 × 片段_时长[1]，因此该机制仅对 8秒延迟目标和 15秒延迟目标生效，如图 7(b) 所示。表 2 显示，重新请求的数量相当可观，这些请求会消耗更多带宽，但通常不会提高体验质量，如图 7(b) 所示。我们注意到，尽管该功能对于原始 L2A-LL 算法在 8秒延迟目标下表现出一定收益（但方差大得多），即使在 3秒目标下也会出现一些重复请求，因为其缓冲区会暂时超过阈值。此外，它只能请求总计 390 个片段中的 124.5 个，这意味着有相当一部分时间处于卡顿状态。

我们建议在低延迟流媒体中避免使用快速切换。

[1]片段_时长因子直到 v4.5.0 才被加入，尽管自 v4.0 起快速切换默认已启用

4.3 早期 dash.js 版本分析

在研究过程中，我们也分析了较早版本的 dash.js。例如，使用 v4.3.0 时，图 7(c) 中的结果（未提供 8 秒结果）显示 Dynamic 带来的体验质量改善更为明显。我们观察到，dash.js v4.5.0 的性能结果不像早期版本那样存在明显差异，不过不同算法的排序保持不变。这些结果凸显了对每个版本进行性能测试的重要性。

5 相关工作

HTTP 自适应流媒体（HAS）系统种类繁多，综述论文中对此有很好的概述 [3, 13]。

已有多项针对低延迟 HAS 系统的研究，主要考察在固定低延迟下单个低延迟自适应码率算法的性能，往往侧重于实现尽可能低的延迟，而未考虑在更高延迟下的性能 [7, 9]。

一项研究 [17] 分析了各种低延迟流媒体播放器，其中包括 dash.js DASH 播放器中的三种自适应码率算法。他们在 Mahimahi [11] 网络仿真器上使用两个 4G 网络配置文件进行测试，并采用单一固定的默认延迟目标（对于 dash.js 为 3 秒）。结果表明，Dynamic 的总体表现最佳；不过，在尽量降低直播延迟方面，LoL 和 L2A-LL 表现更好。L2A-LL 的平均码率优于 LoL，但经历了更多重新缓冲事件。然而，他们所使用网络配置文件的平均码率比测试流中最高码率版本的码率高出一个数量级，这意味着测试难度较低，所探索的网络条件范围也更有限。

目前已有多种用于测试 HAS 系统的方法。Mahimahi [11]系统已被多项研究采用 [10, 17], ，它实现了自己的用户级网络仿真环境，并可通过数据包跟踪文件进行控制。该系统利用 Linux 的隧道接口和网络命名空间来实现会话隔离。会话运行在虚拟机 (VM) 上，并通过虚拟网络接口连接到 Mahimahi 系统。然而，我们的方法采用 Linux 内核流量控制，并通过在独立的物理机器上运行每项实验来实现完整的网络隔离。

Firefox 和基于 Chrome 的浏览器都在开发者工具（DevTools）的网络模块中提供了网络限速功能。网络限速是在应用层执行的，因此无法进行网络层仿真，这意味着它不如我们采用的方法真实。2020 年 ACM 多媒体系统会议的 Grand Challenge [2]

[2]https://2020.acmmmsys.org/lll_challenge.php

该挑战提供了一个基于 Chrome 浏览器网络限速功能的框架，用于提供可控的网络配置文件。这些网络配置文件包含有限的速度和持续时间列表，整体上的吞吐量变化也很少，因此无法很好地代表公共互联网中经历的波动。

6 结论

本文研究了不同延迟目标如何影响 dash.js 播放器的性能和体验质量。我们分析了播放器在一系列真实网络配置文件和不同配置设置下的表现。我们发现，在 dash.js v4.5.0 中，默认的 Dynamic 算法总体上能实现最佳体验质量；不过，LoL 和 L2A-LL 能在更低延迟下实现更高的视频质量，但会导致更多卡顿。与其他算法相比，Dynamic 算法也能更好地遵循目标。

对 L2A-LL 算法表现不佳的进一步分析，使我们得以改进其实现，并取得了显著提升。我们考察了 FastSwitching 算法，发现它在低延迟情况下无法改善总体体验质量，反而浪费带宽。我们的研究还发现，不同软件版本之间的性能可能存在显著差异，这凸显了在部署前、甚至发布前对特定版本进行测试的必要性。

本文有助于根据当前的算法选择适合低延迟流媒体的延迟目标。研究证据表明，追求过低的延迟目标可能会影响体验质量，因此，服务提供商需要在延迟与体验质量之间找到适当的折中。此外，我们通过示例阐明了这一事实：服务提供商需要在部署前，对 dash.js 的具体版本和配置进行充分分析，以确保获得合适的体验质量。

未来，我们希望探索更多改善低延迟流媒体体验质量的方法，包括进一步分析配置和算法改进，并继续探索 L2A-LL 的改进方向。我们还希望研究如何确定每客户端最优延迟目标，使其能够适应客户端的网络状况和自身能力。

参考文献

[1] Chris Bass。2021。10-18。2021 流媒体播放器测试：我们用于测试互联网流媒体的设置。https://www.bbc.co.uk/rd/blog/2021-10-internet-streaming-test-network-conditions-dash.[2] Abdelhak Bentaleb、Mehmet N. Akcay、May Lim、Ali C. Begen 和 Roger Zim- mermann。2022 年。借助 LoL⁺ 捕捉类 Twitch 低延迟直播平台中的精彩瞬间。24（2022 年 1 月），2300–2314。[3] A. Bentaleb、B. Taani、A. C. Begen、C. Timmerer 和 R. Zimmermann。2019 年第一季度。基于 HTTP 的流媒体码率自适应方案综述。*IEEE* 通信调查与教程 21，1 （ 2019 年第一季度），562–585。[4] Giorgos Dimopoulos、Ilias Leontiadis、Pere Barlet-Ros 和 Konstantina Papa- giannaki。2016-11-14，2016 年。从加密流量测量视频体验质量。在*2016* 年互联网测量会议论文集中。计算机协会，美国纽约州纽约市，513–526。[5] Florin Dobrian、Vyas Sekar、Asad Awan、Ion Stoica、Dilip Joseph、Aditya Ganjam、Jibin Zhan 和 Hui Zhang。2011-08-15，2011 年。了解视频质量对用户参与度的影响。在*ACM SIGCOMM 2011* 会议论文集（*SIGCOMM '11*）中。计算机协会，美国纽约州纽约市，362–373。[6] DASH Industry Forum。2021-02-22，2021 年。Dash-Industry-Forum/Dash.Js。https://github.com/Dash-Industry-Forum/dash.js。

[7] Theo Karagkioules、Rufael Mekuria、Dirk Griffioen 和 Arjen Wagenaar。2020-05-27，2020 年。用于低延迟自适应流媒体传输的在线学习。在第 *11* 届 *ACM* 多媒体系统会议论文集（*MMSys '20*）中。计算机协会，美国纽约州纽约市，315–320。[8] S. Shunmuga Krishnan 和 Ramesh K. Sitaraman。2012-11-14，2012 年。视频流质量影响观众行为：使用准实验设计推断因果关系。在*2012* 年互联网测量会议论文集（*IMC '12*）中。计算机协会，美国纽约州纽约市，211–224。[9] May Lim、Mehmet N. Akcay、Abdelhak Bentaleb、Ali C. Begen 和 Roger Zim- mermann。2020-05-27，2020 年。当他们走向高码率时，我们走向低码率：在 Dash.Js 中使用 LoL 实现低延迟直播。在第 *11* 届 *ACM* 多媒体系统会议论文集（*MMSys '20*）中。计算机协会，美国纽约州纽约市，321–326。[10] Hongzi Mao、Ravi Netravali 和 Mohammad Alizadeh。2017 年。使用 Pensieve 实现神经自适应视频流传输。在*ACM* 数据通信特别兴趣组会议论文集（*SIGCOMM '17*）中。计算机协会，美国洛杉矶，197–210。[11] Ravi Netravali、Anirudh Sivaraman、Somak Das、Ameesh Goyal、Keith Winstein、James Mickens 和 Hari Balakrishnan。2015 年。Mahimahi：针对 HTTP 的精确记录与重放。在*2015* 年 *USENIX* 年度技术会议（*USENIX ATC 15*）中。USENIX 协会，美国圣克拉拉，417–429。[12] Werner Robitza、Steve Göring、Alexander Raake、David Lindegren、Gunnar Heikkilä、Jörgen Gustafsson、Peter List、Bernhard Feiten、Ulf Wüstenhagen、Marie-Neige Garcia、Kazuhisa Yamagishi 和 Simon Broom。2018 年。使用 ITU-T 建议 P.1203 进行 HTTP 自适应流媒体 QoE 估计：开放数据库与软件。在第 *9* 届 *ACM* 多媒体系统会议中。阿姆斯特丹。[13] Y. Sani、A. Mauthe 和 C. Edwards。2017 年第四季度。自适应比特率选择：综述。*IEEE* 通信调查与教程 19，4（ 2017 年第四季度），2985–3014。[14] Kevin Spiteri、Ramesh Sitaraman 和 Daniel Sparacio。2018-06-12，2018 年。从理论到实践：改进 DASH 参考播放器中的码率自适应。在第 *9* 届 *ACM* 多媒体系统会议论文集（*MMSys '18*）中。计算机协会，美国纽约州纽约市，123–137。[15] Kevin Spiteri、Rahul Urgaonkar 和 Ramesh K. Sitaraman。2016-04-01，2016 年。BOLA：在线热视频的近最优码率自适应。在*IEEE INFOCOM 2016*——第 *35* 届 *IEEE* 国际通信会议中。IEEE 出版社，美国旧金山，1–9。[16] Gang Yi、Dan Yang、Abdelhak Bentaleb、Weihua Li、Yi Li、Kai Zheng、Jiangchuan Liu、Wei Tsang Ooi 和 Yong Cui。2019 年。ACM Multimedia 2019 直播视频流媒体传输大挑战。在第 *27* 届 *ACM* 国际多媒体会议论文集（*MM '19*）中。计算机协会，美国纽约州纽约市，2622–2626。[17] Bo Zhang、Thiago Teixeira 和 Yuriy Reznik。2022 年。移动网络中低延迟 DASH 和 HLS 流媒体传输的性能。*SMPTE* 运动成像期刊 131，7（2022），26–34。