Camel：视频码率低于目标情况下低延迟直播的帧级带宽估计

刘黎明 Wei Zhang 严乐举 沙洲 Xinggong Zhang Zhidong Jia 谢兰 严冰 Wei Yang 北京大学 蒋力 Feng Qian 马强 班逸轩 北京大学 字节跳动有限公司 字节跳动有限公司 字节跳动有限公司

arXiv:2602.09500v1 [eess.IV] 10 Feb 2026

摘要

低延迟直播（LLS）已成为一种广受欢迎的 Web 应用，许多平台采用 WebRTC 等实时协议，以尽可能降低端到端延迟。然而，我们观察到一种反直觉现象：即使实际编码比特率未充分利用可用带宽，卡顿事件仍然频繁发生。这种带宽利用率不足源于实时视频编码固有的时间变化，导致传统的分组级拥塞控制算法错误估计可用带宽。当突然产生高码率帧时，以错误的速率发送可能触发分组丢失或增加排队延迟，进而导致播放卡顿。

为解决这些问题，我们提出了 Camel，这是一种专为 LLS 设计的新型帧级拥塞控制算法（CCA）。我们的核心思路是利用帧级网络反馈来捕捉真实的网络容量，不受编码导致的不规则发送模式影响。Camel 包含三个关键模块：**带宽与时延估计器**和**拥塞检测器**共同确定平均发送速率，而**突发长度控制器**则负责控制发送模式，以防止数据包丢失。

我们通过大规模实际部署和受控仿真对 Camel 进行了评估。在覆盖 150+ 个国家、拥有 2.5 亿用户和 20 亿个会话的实际平台上，Camel 使 1080P 分辨率占比最高提升了 70.8%，媒体码率提升了 14.4%，卡顿率最高降低了 14.1%。在低于目标、浅缓冲区和网络抖动条件下的仿真中，Camel 的表现优于现有拥塞控制算法：比特率最高提升 19.8%，卡顿率降低 93.0%，带宽估计准确率提升 23.9%。

CCS 概念

• **网络** → **网络算法**; • **信息系统**→ **多媒体流媒体**.

关键词

低延迟直播、拥塞控制、帧级控制、大规模部署



本作品采用知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议进行许可。
WWW '26，阿联酋迪拜© 2026 版权由所有者/作者持有。ACM ISBN 979-8-4007-2307-0/2026/04 https://doi.org/10.1145/3774904.3792535

ACM 参考文献格式：
刘黎明、Zhidong Jia、蒋力、Wei Zhang、谢兰、Feng Qian、严乐举、BingYan、马强、沙洲、Wei Yang、班逸轩和Xinggong Zhang。2026年。Camel：视频码率低于目标时的低延迟直播帧级带宽估计。载于*2026*年*ACM Web*会议论文集（*WWW '26*），*2026*年*4*月*13*日至*17*日，阿联酋迪拜。ACM，美国纽约州纽约市，11页。https://doi.org/10.1145/ 3774904.3792535

1 引言

随着互动式直播成为网络内容交付的主流形式之一 [26]，，对低时延直播（LLS）的需求激增，使其成为当今网络生态系统中的一项关键技术 [25]。端到端 LLS 系统通常由两个核心部分组成：从主播到中继服务器的上行传输，以及从服务器到最终用户的下行分发。流媒体延迟、播放流畅度和视频质量是 LLS 最关注的问题。为降低时延，越来越多的 LLS 服务提供商将上行链路切换为 WebRTC 或 SRT 等实时协议 [12, 27, 35]。

然而，我们在全球最大的视频流媒体平台之一观察到一种出乎意料、违背直觉的现象。如图 2 所示，对约 20 亿个会话的真实数据表明，大多数上传的直播流所使用的带宽不足估计上行带宽的 80%。即便如此，卡顿率仍然很高：超过 5% 的主播，其卡顿率高于 10%。更令人惊讶的是，在卡顿率超过 10% 的跟踪数据中，视频比特率通常远低于估计带宽。

为什么即使带宽未被充分利用，卡顿率仍然居高不下？为方便起见，我们将实际发送比特率持续低于目标比特率或估计的可用带宽的现象称为**低于目标**，并使用视频**编**码比特率与其**目**标比特率之比（ETR）来衡量其严重程度，因为目标比特率通常与估计带宽保持一致。

推理链条虽长，却颇具洞察力。如图1所示，根本原因在于 RTC 视频流量本身的特性，它违背了大多数拥塞控制算法所依赖的持续积压这一基本假设。当发送端始终有足够的数据包等待传输时，测得的网络信号仅受网络状况影响，因而能够准确反映可用带宽。然而，实时编码器以逐帧处理方式运行，帧大小会随时间显著变化，具体取决于内容复杂度和帧类型。虽然编码器旨在使平均比特率与估计带宽相匹配，而非维持恒定的高速率，但其瞬时比特率波动往往导致发送端

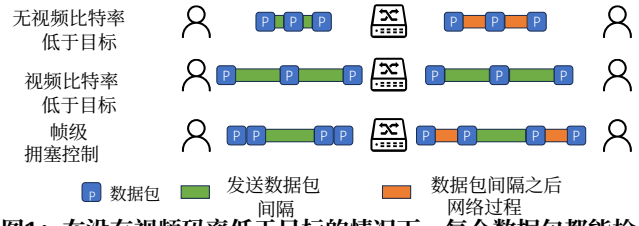


图1：在没有视频码率低于目标的情况下，每个数据包都能检测到网络信息；当视频码率低于目标时，反馈信号也会受到码率欠载的影响，从而导致估计失真；采用帧级拥塞控制时，可以在单个帧突发内正确测量部分网络信息。

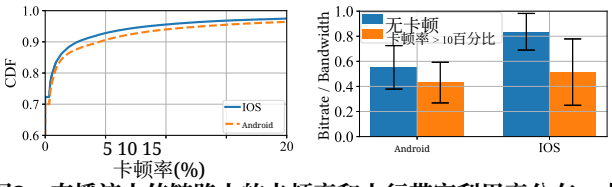


图2：直播流上传链路上的卡顿率和上行带宽利用率分布。大多数直播流仅利用了估计的上行带宽的一部分，但卡顿率仍然很高。此外，在出现卡顿的轨迹中，视频比特率通常远低于估计的上行带宽。

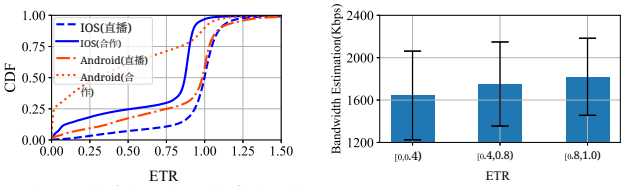
频繁排空发送缓冲区。这会导致比特率欠冲，从而扭曲网络反馈信号：观测到的丢包率和到达间隔不再纯粹反映网络容量，也会受到应用层动态的影响。这种失真会造成不准确的带宽估计。当编码器突然生成一连串高码率帧时，错误的带宽估计可能带来严重后果。如果发送端传输过于激进，数据包会被丢弃，传输延迟也会累积。如果传输过于保守，排队延迟则会增加。在这两种情况下，一旦延迟超过播放缓冲区，就会导致可见的卡顿事件。

那么，我们能否设计一种拥塞控制机制，既能适应编码器自然且不规则的帧级突发模式，又能利用这一模式更准确地估计可用带宽？答案是肯定的。**我们的核心思路是以短促的突发模式而非连续流的方式传输视频数据，然后利用** 帧级网络反馈信号**捕获真实的网络容量，从而不受不规则帧间发送波动的影响**。

尽管前景可期，帧级拥塞控制也带来了若干独特的挑战。首先，在欠估条件下识别可靠的帧级指标并将其准确映射为带宽估计值并非易事。其次，由于相邻帧的延迟可能相互干扰，在帧级检测拥塞十分困难。第三，控制长度

每次传输突发的长度至关重要。较长的突发可提供更多样本，从而实现稳定的带宽估计，但突发过长可能会压垮较小的网络缓冲区，导致分组丢失。

为应对这些挑战，我们提出了 Camel，这是一种专为 LLS 上行视频设计的新型帧级拥塞控制算法。



(a) 实际比特率与目标比特率之比的分布。 (b) 不同低于目标程度下的估计带宽。

图3：RTC上行视频流中码率欠载的分布及影响。

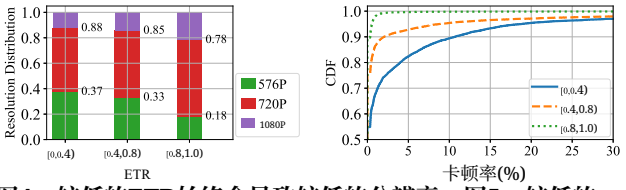


图4：较低的ETR始终会导致较低的分辨率。图5：较低的ETR始终会导致较高的卡顿率。传输。Camel由三个关键模块组成：**带宽与时延估计器**和**拥塞检测器**共同确定平均发送速率，而**突发长度控制器**则控制数据包发送模式，以避免分组丢失。

值得一提的是，Camel 的设计灵感源自分组列车概念，并采用类似的方法估算网络带宽，实现帧级拥塞控制。然而，与以往关于分组列车的研究 [4, 8–10, 14, 16, 18, 23, 24, 28, 31–33], 不同，Camel 提出了一种专为 LLS 设计的新型拥塞控制算法。具体而言，它提出了在途数据-延迟拥塞检测方法（§4.2），有效减少了拥塞误判，并设计了突发长度控制器（§4.3），以防止帧级突发导致的分组丢失。此外，它还在 §4.1 中提出了一种带宽指示器。

我们在大规模真实场景和受控模拟环境中对系统进行了广泛测试。在真实场景测试中，我们将系统部署在全球最大的直播平台之一，为150+ 个国家的2.5亿用户提供服务，处理了20亿次会话；结果显示，1080P分辨率占比最高提升70.8%，媒体码率提升14.4%，卡顿率最高降低14.1%。在模拟环境中，我们评估了该算法在视频比特率低于目标、浅缓冲区和网络抖动条件下的性能。结果表明，Camel显著优于现有的拥塞控制算法，比特率最高提升19.8%，卡顿率降低93.0%，带宽估计准确度最高提升23.9%。

本文其余部分安排如下：第 2 节介绍本研究的背景与动机。第 3 节概述 Camel，包括其关键组件和数据流。第 4 节详细介绍 Camel 的设计。第 5 节评估我们的系统，随后在第 6 节讨论相关工作。最后，第 7 节总结全文。

2 背景与动机

2.1 常见的低于目标原因导致体验质量低下

正如从全球最大的视频流媒体平台之一收集的真实世界数据所示，我们观察到，低于目标行为普遍存在：实际发送比特率持续低于

目标比特率或估计的可用带宽，并广泛存在于基于 RTC 的上行会话中。图3(a)展示了在 12 秒间隔内，实际编码比特率与用户估计的上传带宽之比的分布。结果表明，超过 50% 的流呈现出不同程度的低于目标现象。这一普遍现象可归因于实时视频编码固有的若干因素。首先，实时编码器通常旨在使平均比特率与估计的可用带宽相匹配，而不是持续保持较高的速率，因为视频内容和帧类型会随时间发生显著波动 [13]：I 帧的大小可能超过 P 帧或 B 帧的 10 倍，导致除 I 帧之外的大多数时间里编码器都有未使用的带宽。其次，为了保持低延迟并防止过度排队，实时编码器会频繁降低瞬时比特率，进一步导致持续的低于目标现象 [12]。

为了解低于目标为何重要，接下来我们考察其对带宽估计的影响。图 3(b) 展示了按不同码率与带宽比值分组的 LLS 流的估计带宽。由于不同分辨率的视频比特率范围各不相同，我们以 720P 流为例。结果表明，随着低于目标的程度加剧，现有拥塞控制算法（CCAs）往往会估计出更低的平均带宽和更高的方差。鉴于编码比特率是视频编码器固有的属性，与传输层估计无关， [37], 这种相关性表明，码率低于目标会直接抑制发送端探测可用容量的能力，进而导致系统性的带宽低估。

这种带宽低估会直接造成两种后果，共同损害用户体验质量（QoE）。图 4 显示，在较低的码率与带宽比值下，720P+ 视频的占比显著下降；图 5 则显示，这些情况下的卡顿率持续上升。综合来看，这些发现表明，码率低于目标不仅普遍存在，还会严重损害 LLS 系统的播放流畅度和用户感知的视频质量。原因在于，这会妨碍发送端正确地控制大型视频帧的发送节奏。当编码器突然生成高码率帧时，发送端面临两个次优选择：如果发送过于激进，数据包会被网络丢弃，传输延迟也会不断累积；如果发送过于保守，排队延迟则会增加。无论哪种情况，一旦延迟超过播放缓冲区，就会出现明显的卡顿事件。此外，在带宽持续被低估的情况下，自适应码率（ABR）算法会受到误导，选择较低的视频分辨率，从而导致画质明显下降 [30]。

2.2 洞见：帧级突发

这种不可靠性的根本原因在于 RTC 视频流量本身的时间结构，它违背了大多数拥塞控制算法所依赖的持续积压这一基本假设。当发送端始终有足够的数据包等待发送时，测得的吞吐量、延迟和丢包率等网络信号仅受网络状况影响。因此，传统的分组级拥塞控制算法在文件下载等大批量数据传输场景中表现出色。在这些场景下，拥塞窗口（拥塞窗口）的动态变化，以及 BBR [2], CUBIC [11], 或 Copa [1] 等算法基于反馈进行的调整，都能紧密

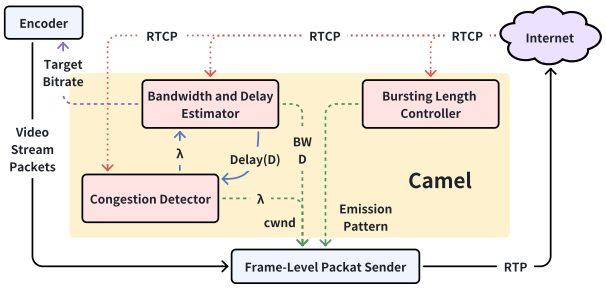


图6：Camel 的系统架构。

跟踪真实的可用带宽。然而，RTC 流量从根本上打破了这一假设。与始终有足够数据包可供发送的文件传输或大批量流媒体应用不同，实时编码器采用逐帧处理方式。每一帧都必须先完成编码才能传输，而且帧大小会随时间显著变化，具体取决于内容复杂度和帧类型。因此，发送端会以短促、不规律的数据包突发进行传输，随后出现空闲间隔。这些由编码器造成的静默期经常会耗尽发送缓冲区，导致码率欠载，进而扭曲网络反馈信号：观测到的丢包率和到达间隔不再纯粹反映网络容量，也受到应用层动态的影响。

受这一观察启发，我们提出这样一个问题：能否设计一种拥塞控制机制，既与编码器自然且不规则的帧级突发模式相契合，又能利用这一模式更准确地估计可用带宽？

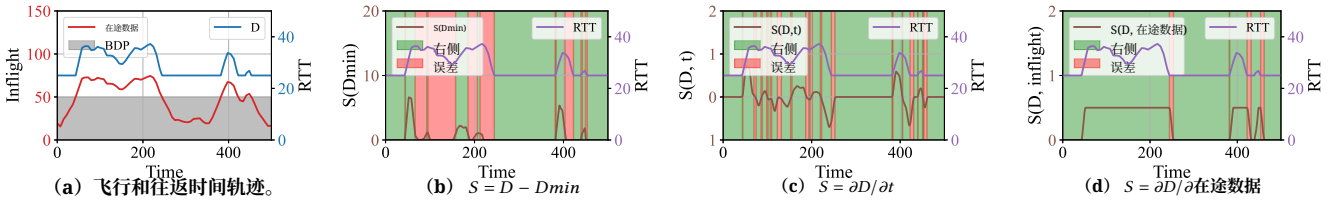
我们的答案是肯定的。如图1所示，核心思路是将视频数据分成短促的突发片段进行传输，而不是持续不断地传输。随后，我们利用帧级网络反馈 来估计可用带宽，该反馈会汇总一帧内的网络信号。通过基于帧级统计信息进行估计，即使发送模式不规则或突发之间存在空闲间隔，估计结果也不会受到影响，从而使拥塞控制器能够准确捕捉真实的网络容量，即使在码率低于目标时也是如此。

2.3 挑战

尽管帧级拥塞控制算法前景可期，但它也带来了一些必须谨慎应对的独特挑战。

挑战1：根据帧级反馈估计带宽和延迟。在低于目标的情况下，识别可靠的帧级指标并将其映射为准确的带宽估计值并非易事。帧级可用的网络反馈信号可能会受到帧大小和编码速率变化的干扰。现有的帧级拥塞控制算法，例如依赖帧传输带宽的 SQP [24], 和使用带宽利用率（BUR）的 Pudica [32], , 都会受到这一问题的影响。当帧小于预期的满尺寸帧时，这些算法会根据低于目标的比例进行补偿，从而放大较小帧中已有的统计误差，导致带宽估计不稳定或存在偏差。

挑战2：基于帧级时延的拥塞检测。 现有的数据包级拥塞信号用于帧级控制时存在若干缺点。具体而言，



（a）飞行和往返时间轨迹。 （b） $S = D - Dmin$ （c） $S = \partial D/\partial t$ （d） $S = \partial D/\partial$在途数据

图7：不同拥塞信号的准确性，其中 $\partial D/\partial$在途数据能够实现最准确的结果。

相邻帧的时延很容易相互干扰，因此难以在帧粒度上分离拥塞效应。

基于丢包的算法 [11] 在发生物理丢包时往往吞吐量较低，而且无法有效控制深缓冲区网络中的时延。基于 MinRTT 探测的算法 [2], 将观测到的最小 RTT 作为传播时延的估计值，在帧级场景下也不可靠。用于跟踪最小 RTT 的窗口过大，就无法适应网络抖动或动态变化的背景流；窗口过小，则在出现拥塞后可能导致排队时延无法收敛。GCC 采用的时延梯度法 [17], 对网络抖动的适应性更强，但在稳定下来后无法主动消除持续排队时延。此外，按时间排序的时延梯度依赖于数据包大小一致、发送节奏速率固定等假设，而这些假设在帧级拥塞控制中并不成立，因为其中的数据包速率和帧大小都会大幅变化。

挑战 3：突发长度控制。由于网络基础设施和运营商策略 [9] 存在差异，网络的缓冲区容量也各不相同。因此，必须谨慎控制每次传输突发的长度。较长的突发长度使算法能够在每次突发中纳入更多数据包样本，从而提高带宽估计的稳定性。然而，突发过大很容易使小型缓冲区不堪重负，导致数据包丢失，且丢包通常发生在突发末尾。因此，必须根据网络环境自适应地配置最大突发长度，在带宽估计的准确性与缓冲区溢出风险之间取得平衡。

3 Camel 概述

为应对上述挑战，我们提出了 Camel，这是一种专为 LLS 上行视频传输设计的新型帧级拥塞控制算法。拥塞控制器必须做出两个基本决策：（1）发送窗口（拥塞窗口）应有多大；（2）采用何种数据包发送模式。此外，现代拥塞控制算法必须确保发送窗口与网络能够容纳的数据包数量相匹配，同时快速响应拥塞信号 [2, 11]。因此，发送窗口通常根据以下估计值设定：

$$\text{cwnd} = \gamma \cdot \hat{\text{BDP}}, \tag{1}$$

ˆ其中带宽时延积表示估计的带宽时延积（带宽时延积），γ 是一个动态缩放因子，它根据观测到的拥塞进行自适应，以在利用率与排队延迟之间进行权衡。在 Camel 中，我们将这些决策分解为三个模块：**带宽与时延估计器**（用于ˆ带宽时延积，§4.1）和**拥塞检测器**（用于 γ，§4.2）共同确定拥塞窗口，而**突发长度控制器**（§4.3）则控制数据包发送模式。每个模块都直接针对§2.3 中指出的某一项挑战。

图6展示了 Camel 的整体架构：**带宽与时延估计器**持续分析传入的 RTCP 反馈，以同时估计可用带宽和延迟。随后，估计带宽被映射为编码器的目标比特率，从而在应用层指导速率控制，并同时传递给**数据包发送器** 模块。估计延迟被转发至**拥塞检测器**，后者结合时延趋势与 RTCP 反馈，推导出反映实时网络状态的拥塞信号。与此同时，**突发长度控制器** 根据估计的网络状况，为下一个传输周期确定合适的突发长度。最后，数据包发送器利用这三个模块的输出（估计带宽、延迟、拥塞状态和突发长度）来调度来自编码器的数据包传输，使其模式与帧级突发对齐，同时遵守网络约束。

4 Camel 的设计

4.1 带宽与时延估计器

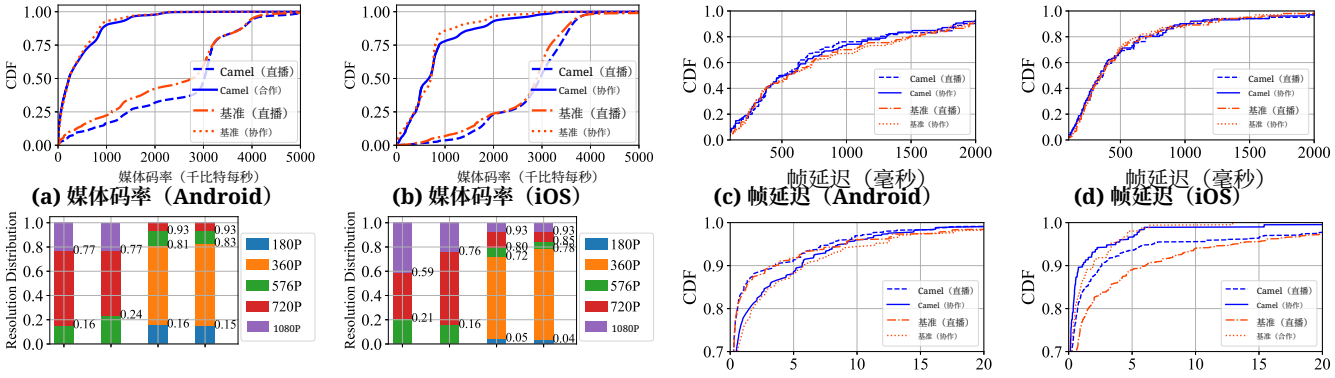
为应对挑战1，带宽与时延估计器模块以实时传输控制协议反馈为输入，并输出三个关键的帧级指标：带宽、延迟和带宽时延积。

帧级带宽估计。 受包列算法 [9, 14] 和 Salsify [10], 的启发，我们的核心思路是：通过连续快速发送每帧的数据包，将数据包间隔降至最小，接收端便能根据这些数据包的到达速率准确推断可用网络带宽。对于由 n 个数据包组成的一帧，这些数据包的编号从 1 到 n，帧级带宽 B 定义如下：

$$B = \frac{\sum_{i=2}^{n} S_i}{t_{recv_n} - t_{recv_1}}, \tag{2}$$

其中，S_i 表示第 i个数据包的大小，t_{recv_i} 表示其接收时间。由于该带宽是针对每一帧独立计算的，因此发送端无需在应用层持续积压数据包。

帧级时延估计。为减轻突发传输所导致的自引入时延失真，我们将每帧第一个数据包的往返时延（RTT）定义为帧级时延 D。与以固定间隔发送数据包的传统 pacing 机制不同，我们的帧级方法会以突发方式快速发送一帧内的所有数据包。这种突发传输模式可能会暂时使网络缓冲区饱和，造成无法准确反映真实网络拥塞状态的排队时延。由于突发中的后续数据包会引发排队延迟，只有第一个数据包不受其影响，因此其 RTT 能够更可靠地反映底层网络状况。



(a) 媒体码率（Android） **(b) 媒体码率（iOS）** **(c) 帧延迟（Android）** **(d) 帧延迟（iOS）**

(e) 分辨率（Android） **(f) 分辨率（iOS）** **(g) 卡顿率（Android）** **(h) 卡顿率（iOS）**

图 8：……的结果　　大规模 A/B 测试表明，Camel 在不增加时延的情况下提升了视频体验质量　　.

帧级带宽时延积。借助估算得到的帧级带宽和时延，我们可以进一步推导出帧级带宽时延积（BDP），所依据的原理与经典拥塞控制模型 [2] 中的类似：

$$\hat{BDP} = \text{avg}(B) \times \min(D), \tag{3}$$

其中，avg(B) 表示平均帧级带宽，反映实际可用的网络容量；min(D) 表示观测到的最小帧级时延，可近似视为往返传播时间（RTprop）。

4.2 拥塞检测器

为应对挑战2并实现稳健的帧级拥塞检测，Camel 将帧级时延变化相对于在途数据量的短期趋势作为信号。具体而言，我们计算帧级时延相对于在途字节数的梯度。如果该梯度超过预设阈值，则判定发生拥塞，因为该梯度反映了发送端向网络注入更多数据时排队延迟的增长情况。本节余下部分将首先通过实验说明选择这一拥塞信号的理由，然后介绍如何将其映射为拥塞窗口缩放因子，最后定义向应用层报告的带宽值。**拥塞信号的依据**。为了比较我们提出的拥塞信号与几种常用检测方法的准确性，我们生成了一条合成轨迹。我们假设网络配置固定，带宽为 2 Mbps，往返传播时延（RTprop）为 25 毫秒，因此带宽时延积（BDP）为 50 Kb。随后，我们生成了一系列在途数据量，以模拟编码器的比特率波动，并按如下方式计算每个数据包的往返时延（RTT）：

$$\text{RTT} = \begin{cases} \text{RTprop} + \frac{\text{inflight} - \text{BDP}}{\text{Bandwidth}}, & \text{if inflight} > \text{BDP}, \\ \text{RTprop}, & \text{otherwise.} \end{cases} \tag{4}$$

该表述遵循典型的排队时延模型：当在途数据量未超过网络容量时，往返时延保持在传播延迟不变；而一旦超过带宽时延积，往返时延便线性增加。基于该模型，我们将往返时延超过往返传播时间的任意时刻定义为真实拥塞状态。

我们将提出的基于在途数据的时延梯度信号,$S(D,$ 在途数据$) = \partial D/\partial$在途数据，与两种常用替代方案进行了比较：基于最小往返时延的排队信号，$S(D_{\min}) = D - D_{\min}$，以及基于时间的时延梯度，$S(D,\ t) = \partial D/\partial t$。为确保比较公平，所有方法均采用一致的 5 秒观测窗口。

如图7所示，高亮区域标示了各方法正确或错误识别拥塞的时间间隔。对于另外两种信号，误差出现在队列已经形成积压之后，而我们基于在途数据的延迟梯度始终能取得显著高于基线方法的总体准确率。**将拥塞信号映射到窗口缩放**。为根据检测到的拥塞控制发送窗口，我们将 γ 的初始值设为1。当检测到拥塞时，将 γ 乘以0.95以减小，从而主动缩小拥塞窗口并缓解潜在的队列积压：

$$\gamma_{t+1} = \gamma_t \times 0.95, \tag{5}$$

若未检测到拥塞，则将 γ 重置为1，使拥塞窗口恢复至完整大小并充分利用可用带宽。

拥塞调整带宽上报。为向应用层提供更准确的可用网络容量视图，Camel向应用层上报拥塞调整带宽。所上报的目标比特率计算如下：

$$\text{target bitrate} = \gamma \times \text{avg}(B), \tag{6}$$

其中 γ 是由拥塞检测得到的窗口缩放因子，avg(B)是平均帧级带宽。

4.3 突发长度控制器

为应对挑战 3，Camel 引入了突发长度控制器，旨在不造成分组丢失的前提下最大化突发长度。我们根据观测到的丢包率推断网络缓冲区长度，因为我们发现：对于深缓冲区，分组丢失主要由物理因素造成，表现为不同大小的帧具有相近的丢包率。相反，对于浅缓冲区，分组丢失主要由缓冲区溢出导致，因此较大的帧丢包率更高。具体而言，我们将帧切分为固定的 2KB 间隔，并分别计算每个间隔的丢包率。最小间隔的最低丢包率 L_0 可作为物理丢包率的估计值。缓冲区长度估计 M 每 5 秒更新一次：

$$M_t = \begin{cases} M_{t-1} - 2\ \text{KB}, & \text{if } L_i > L_0 + 0.1 \\ M_{t-1} + 2\ \text{KB}, & \text{otherwise} \end{cases} \quad (7)$$

其中，L_i 指当前间隔的丢包率。如果当前间隔的丢包率超过 L_0 的 10%，则将 M 减小；否则，将 M 增大。如果 M_t 已达到其最小值，而丢包率仍高于 L_0，算法则退回到基于 GCC 的方法。一旦估算出缓冲区长度 M，便会在周期性帧发送算法中用它限制最大帧长度，从而降低缓冲区溢出的风险。

5 评估

本节从五个关键方面评估 Camel：真实场景中的大规模 A/B 测试（§5.1）；视频比特率低于目标条件下的鲁棒性（§5.2）；弱网环境中的性能（§5.3）；用于分离并理解各项贡献的组件分析（§5.4）；以及与背景流量竞争时的收敛行为（§5.5）。

设置：我们的大规模实验在全球最大的视频流媒体平台之一上进行，该平台服务于 150 多个国家和地区的 2.5 亿多用户，累计会话量达 20 亿次。考虑到在生产环境中部署新算法需要谨慎，我们的大规模 A/B 测试仅将平台现有的拥塞控制算法（基于 BBR 的变体）作为对照。其他所有实验均在自行搭建的软件仿真环境中进行。

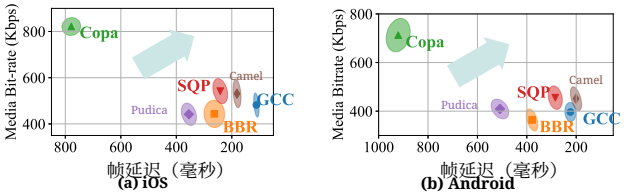
基线方法：在仿真实验中，我们将 Camel 与六种基线算法进行比较：GCC [3], BBR [2], Copa [1], Pudica [32], SQP [24], 和 Salsify [10]。由于 Salsify 需要功能性编码器，因此无法集成到我们的流媒体系统中。因此，我们使用其开源实现进行评估。由于处理流水线存在显著差异，Salsify 的帧延迟无法直接比较，因此未纳入结果。此外，由于 Salsify 没有明确的目标码率，我们未在码率欠载实验中纳入该算法。

指标：我们使用四项关键指标评估 Camel：媒体码率、帧延迟、卡顿率和带宽估计准确度。媒体码率用于量化有效媒体传输速率，反映质量和效率。帧延迟衡量视频帧的端到端时延。卡顿率表示
播放时间中处于卡顿状态的比例，其中卡顿时长定义为服务器记录的超过 200 毫秒的帧间隔。带宽估计准确度用于评估报告给应用层的带宽与实际吞吐量的接近程度，从而确保对不同算法进行公平比较。

5.1 大规模真实世界 A/B 测试

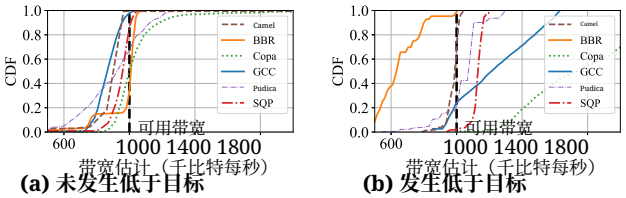
总体而言，大规模 A/B 测试表明，Camel 在不影响时延的情况下提升了视频质量和流畅度。
媒体码率。如图 8(ab) 所示，Camel 的媒体码率始终高于基线。对于 Android 用户，Camel 在直播中的平均码率提升了 11.9%，在联合直播中提升了 7.6%。对于 iOS 用户，提升幅度分别达到 3.5% 和 14.4%。

分辨率。图 8(ef) 显示，Camel 利用更高的比特率提升了视频分辨率。在 Android 上，分辨率达到或高于 720P 的视频占比提高了 10.5%。在 iOS 上，直播中 1080P 视频的占比



(a) iOS (b) Android

图 9：在低于目标的仿真场景中，Camel 在比特率与帧延迟之间实现了更好的权衡。



(a) 未发生低于目标 **(b) 发生低于目标**

图10：Camel 能够在视频比特率低估场景下准确估算带宽。

从 24% 上升至 41%，联合直播中分辨率至少达到 576P 的视频占比提高了 27%。**卡顿率**。图 8(gh) 显示，Camel 整体上实现了更低的卡顿率。在 Android 上，差异较小；而在 iOS 上，直播的平均卡顿率下降了 14.1%。在联合直播中，平均卡顿率基本相近，但长尾卡顿事件显著减少，从而改善了极端情况下的用户体验。**帧延迟**。如图 8(cd) 所示，Camel 在 Android 和 iOS 上的帧延迟均与基线相当，同时显著提升了比特率和稳定性。

填充码率。基线算法在直播中将平均填充码率维持在至少 5 千比特每秒，在联合直播中维持在至少 25 千比特每秒；而 Camel 将填充码率降至接近零，因为只有在探测非活跃连接时才会进行填充。

5.2 欠载条件下的鲁棒性 g

整体性能。我们评估了所有算法在视频比特率低于目标条件下的整体性能。我们根据图3(a)所示的联合直播场景分布，为 iOS 和 Android 模拟了两组码率欠载轨迹。可用带宽上限设为 1000 千比特每秒，每种算法运行 150 秒。由于未观察到明显卡顿，我们重点关注媒体码率和帧延迟。

如图 9 所示，除 Copa 外，Camel 的比特率均高于大多数基线方法。具体而言，与 GCC 相比，Camel 在 Android 上将平均比特率提高了 4.7%，在 iOS 上提高了 19.8%。这一提升源于即使编码器输出低于目标，Camel 仍能准确估计可用带宽。相比之下，BBR 持续低估带宽，而 Pudica 因假设帧大小稳定而表现出较大的波动。Copa 倾向于高估带宽，导致帧延迟过长，难以及时恢复。**带宽估计**。我们进一步考察了欠载条件下的带宽估计准确率。可用带宽固定为 1000 千比特每秒，每种算法运行 90 秒。在 30 秒至 60 秒期间，我们手动将编码比率降低至 60%，以模拟低于目标的情况。我们将此期间的估计带宽与正常阶段的估计结果进行了比较，如图 10 所示。在基线方法中，只有帧级算法 SQP 和 Pudica 能够在低于目标的情况下估计带宽。由于接收速率降低，BBR 出现严重的带宽低估；而 GCC 和 Copa 等基于延迟的方案在未观察到时延累积时会高估带宽。相比之下，得益于帧级设计，Camel 即使在帧大小变化时仍能保持稳健，因此兼具更高的准确率和稳定性。具体而言，Camel 的平均估计准确率达到 98.9%，比 GCC 高出 23.91%。

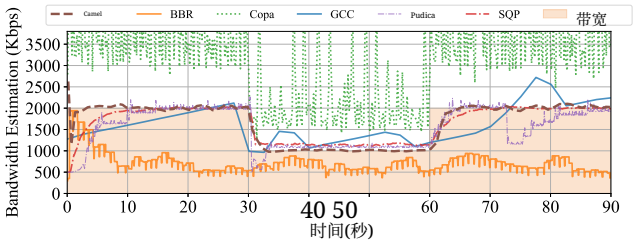


图11：在视频比特率低于目标的场景下，Camel 能够正确跟踪带宽波动。

我们将此期间的估计带宽与正常阶段的估计结果进行了比较，如图 10 所示。在基线方法中，只有帧级算法 SQP 和 Pudica 能够在低于目标的情况下估计带宽。由于接收速率降低，BBR 出现严重的带宽低估；而 GCC 和 Copa 等基于延迟的方案在未观察到时延累积时会高估带宽。相比之下，得益于帧级设计，Camel 即使在帧大小变化时仍能保持稳健，因此兼具更高的准确率和稳定性。具体而言，Camel 的平均估计准确率达到 98.9%，比 GCC 高出 23.91%。

响应能力。 我们评估了各算法在码率低于目标时对带宽变化的响应能力。每种算法运行 90 秒：前 30 秒带宽为 2000 千比特每秒，中间 30 秒降至 1000 千比特每秒，最后 30 秒恢复至 2000 千比特每秒。图 11 展示了随时间变化的带宽估计值。包括 Camel、SQP 和 Pudica 在内的帧级算法正确跟踪了带宽波动。然而，由于在码率低于目标时接收速率持续下降，BBR 的估计值一直偏低。基于延迟的算法 GCC 和 Copa 高估了带宽，因此无法很好地响应实际网络变化。

5.3 弱网络下的性能

真实世界 4G/5G 网络轨迹下的性能。 我们使用公开数据集中的 4 条真实世界长期演进网络轨迹 [22]，，评估了 Camel 在多种网络类型和移动性条件下的表现。表1显示，与其他帧级方法（SQP、Pudica、Salsify）相比，Camel 可将卡顿率降低 13%–49%。Salsify 因编码器资源占用较高而帧率较低；Pudica 假设帧大小恒定；SQP 则收敛缓慢，导致带宽骤降时出现卡顿。与 GCC 相比，Camel 在 4G 网络轨迹上的比特率最高提升 52%，这是因为其基于数据包序列的设计能够快速响应带宽增加。尽管 BBR 和 Copa 的吞吐量较高且卡顿率较低，Camel 仍可将帧延迟降低 12%–17%，避免周期性的缓冲区填充开销。

应对网络抖动的性能。图18展示了在带宽为 1000 千比特每秒、存在网络抖动时 Camel 的表现。BBR、SQP 和 Pudica 依赖最小往返时间探测，会将瞬时抖动误判为持续拥塞，导致带宽低估、前向纠错传输不足、严重卡顿以及较高的帧延迟。Copa 通过过滤极端往返时延来缓解抖动，其比特率高于采用最小往返时间探测的方法，但其保守的更新策略会延迟拥塞消解，导致卡顿持续发生。GCC 使用延迟梯度，在卡顿极少的情况下保持较低的帧延迟。Camel 通过基于在途帧而非按时间顺序排列的数据包计算延迟梯度，进一步提升了鲁棒性，降低了抖动的影响，其比特率最高比 GCC 高 94.9%。

应对分组丢失的性能。图19展示了瓶颈带宽为 1000 千比特每秒时，在分组丢失情况下的性能。随着丢包率上升，所有算法的比特率都会因有效容量降低而下降。高丢包率下，GCC 和 BBR 严重低估带宽，影响重传和前向纠错效率，并导致更多重新缓冲。SQP 和 Pudica 在高丢包率下高估带宽，也会增加重新缓冲。在丢包率为 50% 时，它们的卡顿率分别比 Camel 高 4.53% 和 2.62%。

浅缓冲区场景下的性能。图20比较了缓冲区大小为 2KB 至 10KB 时 Camel 与基线算法的表现。在 2KB 时，SQP 和 Pudica 缺少足够的数据包样本，无法准确进行带宽估计；而 Copa 因排队延迟较低而无法检测到拥塞。这些算法持续过冲，导致严重卡顿和更长的延迟；分组丢失还会进一步降低媒体码率。BBR 和 GCC 则通过权衡码率与延迟，表现良好。Camel 能够检测浅缓冲区，并回退到 GCC，取得相近的性能，且优于其他帧级方法。随着缓冲区长度增加，GCC 因频繁切换模式而性能下降；而 SQP 和 Pudica 分别要到缓冲区大小超过 4KB 和 6KB 时，才能开始准确估计带宽。Camel 仍能通过优化调整帧长度，提供更优的体验质量。

5.4 组件分析

突发长度控制器。图12展示了突发长度控制器的有效性。对于非常浅的缓冲区（2KB），基于分组列车的方法会失效，此时控制器会缩短突发长度，使 Camel 能够回退到 GCC。随着缓冲区长度增加，突发长度也相应增大，在避免缓冲区溢出的同时，最大限度地增加带宽采样器的样本数。在 10KB 时，突发长度超过 12KB，使大多数大型帧能够完整发送，从而降低帧延迟并改善带宽估计。

图13将突发长度控制器与固定的20KB基线进行了比较。在2KB缓冲区中，其媒体码率达到1.79×，并将卡顿减少了67%。对于4–6KB缓冲区，它避免了不必要的分组丢失，减少了重传和前向纠错，并显著提升了体验质量。

5.5 与背景流量的收敛性

我们在多流场景中评估了 Camel 的拥塞检测能力，使用的公共带宽为1000千比特每秒。结果表明，它在面对各种类型的背景流量时都能实现良好的公平性。**流间公平性。** 我们评估了多个往返时延相同的 Camel 流之间的公平性。每隔10秒引入一个新的 Camel 流。如图14所示，Camel 成功实现了公平的带宽分配和收敛的时延。

与非弹性流量的竞争能力。 我们评估了 Camel 与用户数据报协议等非弹性流量共存的能力，以确保它不会持续进行不必要的带宽竞争。实验持续了90秒，期间有一个用户数据报协议流加入。我们收集了吞吐量和分组单向时延。如图15所示，Camel 与用户数据报协议流有效共享带宽，同时保持时延稳定且可控。

与弹性流量共存时的竞争性。随后，我们评估了 Camel 与 GCC 等弹性流量共存的能力，以确保它在与低时延算法竞争时既不会饿死其他流量，也不会被饿死。Camel 运行了 90 秒，期间引入了一条 GCC 流。图16展示了两条流的吞吐量和往返时延轨迹，

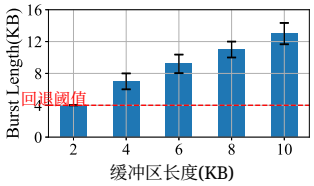
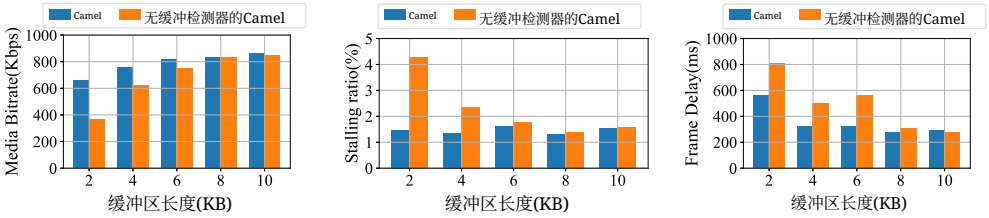


图12：Camel 确定的不同网络缓冲区长度下的突发长度。



(a) 媒体码率 **(b) 卡顿率** **(c) 帧延迟**

图13：与浅缓冲区对抗时，Camel 的体验质量提升情况（采用突发长度控制器）。

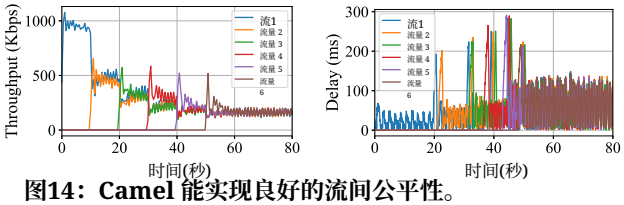


图14：Camel 能实现良好的流间公平性。

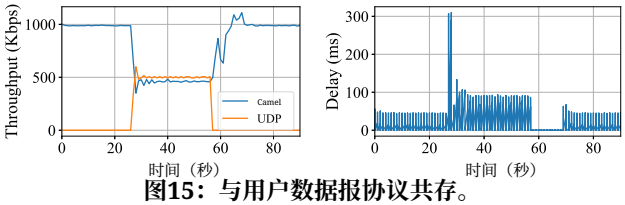


图15：与用户数据报协议共存。

表明 Camel 成功地与 GCC 共享带宽，同时实现了可控的时延。与 TCP CUBIC 等填充缓冲区的流量共存时，Camel 不会激进地争抢带宽，因为过高的时延会对 LLL 流媒体的体验产生负面影响。在浅层网络缓冲区下，Camel 能够维持公平的带宽份额；而在深层缓冲区场景中，让 TCP 快速获得带宽、完成数据传输并退出网络，则可能是一种有效策略。图17展示了一项实验：在 Camel 传输视频流期间引入了一条 TCP CUBIC 流。结果表明，当网络缓冲区设为 500 KB 时，Camel 会维持最低限度的传输速率；而当网络缓冲区减小到 100 KB 时，Camel 与 TCP CUBIC 大致平分带宽。

6 欠估条件下带宽预测的相关工作

应用受限。 Sammy [28]是一种新型自适应码率（ABR）算法，旨在平滑视频流量，使其对互联网应用更加友好。然而，流量平滑后，基于吞吐量的算法难以准确估计带宽，导致 Sammy 与我们的方法一样处于"应用受限"状态。为缓解这一问题，Sammy 会在初始化期间过量传输流量，之后再根据具体的算法需求调整过量传输的程度。不过，如果底层拥塞控制算法能够有效应对应用受限的情况，Sammy 在选择 ABR 算法时就会更加灵活，同时仍能保持流量平滑。

自适应码率与帧率。自适应比特率与帧率自适应是 LLS 的热门课题。Fugu [34] 利用基于真实世界数据训练的深度神经网络，实现稳健的吞吐量预测。Sensei [36] 通过评估用户对视频不同部分的质量敏感度来改进自适应比特率算法。SODA [5]优化三项关键指标：码率切换、比特率与卡顿，同时降低算法复杂度以增强可部署性。ARTEMIS [29]为直播动态配置码率阶梯，以适应随时间变化的视频内容。AFR [21]引入自适应帧率控制，以提升视频流媒体的帧率与分辨率。若拥塞控制算法即便在应用受限条件下也能提供稳定且准确的带宽估计，自适应比特率与帧率自适应便可进一步优化以提升性能。

传输层优化。 除了拥塞控制算法的设计之外，传输层还有其他几种优化方法。AUGUR [38] 利用蜂窝网络上的多路径传输，降低云游戏中由 Wi-Fi 波动引发的长尾时延。Converge [7] 通过纳入视频帧信息来改进多路径算法，从而提高帧传输的完整性。Zhuge [19] 根据网络队列的状态，在接入点（AP）主动拦截确认应答，以加快拥塞控制算法的响应。Hairpin [20] 引入了一种丢包恢复机制，通过优化重传与前向纠错（FEC）的组合，在满足截止期限的同时，为基于边缘的交互式流媒体节省带宽。

编码与传输互联。 已有若干方法探索编码层与传输层之间的协同，以获得更好的性能，但往往以更高的部署复杂度为代价。Swift [6] 采用分层编码与神经视频编解码器，以缓解因网络容量估计不准确而导致的自适应比特率问题。BurstRTC [15, 16] 通过对帧大小分布建模来预测帧延迟和体验质量。Salsify [10] 通过定制编码器将其与传输层集成，从而提升编码器效率。

7 结论

本文针对现有低延迟直播系统上行拥塞控制算法设计中的一个关键瓶颈展开研究：视频码率低于目标的现象十分普遍。我们提出了 Camel，这是一种新颖的帧级拥塞控制算法，可降低对大规模或连续数据的依赖。实验结果表明，Camel 能够在各种网络状况下准确估计带宽，同时显著提高实际部署中的媒体码率并减少重新缓冲。本研究不存在任何伦理方面的顾虑。本研究由国家自然科学基金委员会项目（62431017）和字节跳动项目（CT20241126107484）资助。我们衷心感谢智能媒体技术重点实验室提供的支持。通讯作者为 Xinggong Zhang。

参考文献

[1] Venkat Arun 和 Hari Balakrishnan。2018。Copa：面向互联网的实用{基于延迟}拥塞控制。载于第15届USENIX网络系统设计与实现研讨会（NSDI 18）。329–342。[2] Neal Cardwell、Yuchung Cheng、C Stephen Gunn、Soheil Hassas Yeganeh 和 Van Jacobson。2017。BBR：基于拥塞的拥塞控制。《ACM 通讯》60卷2期（2017），58–66页。[3] Gaetano Carlucci、Luca De Cicco、Stefan Holmer 和 Saverio Mascolo。2017。网页实时通信的拥塞控制。《IEEE/ACM《网络汇刊》》25卷5期（2017），2629–2642页。[4] Dispersions Carry。 [无日期。]。包对离散传递了什么信号？（[无日期]）。[5] Tianyu Chen、Yiheng Lin、Nicolas Christianson、Zahaib Akhtar、Sharath Dharmaji、Mohammad Hajiesmaili、Adam Wierman 和 Ramesh K Sitaraman。2024。SODA：用于实现稳定高质量视频流媒体的自适应码率控制器。载于 ACM SIGCOMM 2024 会议论文集。613–644页。[6] Mallesham Dasari、Kumara Kahatapitiya、Samir R Das、Aruna Balasubramanian 和 Dimitris Samaras。2022。Swift：采用分层神经编解码器的自适应视频流媒体。载于第19届USENIX网络系统设计与实现研讨会（NSDI 22）。103–118页。[7] Sandesh Dhawaskar Sathyanarayana、Kyunghan Lee、Dirk Grunwald 和 Sangtae Ha。2023。Converge：面向 WebRTC 的体验质量驱动型多路径视频会议。载于 2023年ACM SIGCOMM会议论文集。637–653页。[8] Constantinos Dovrolis、Parameswaran Ramanathan 和 David Moore。2004。包离散技术与容量估算方法。《IEEE/ACM《网络汇刊》》12卷6期（2004），963–977页。[9] Tobias Flach、Pavlos Papageorge、Andreas Terzis、Luis Pedrosa、Yuchung Cheng、Tayeb Karim、Ethan Katz-Bassett 和 Ramesh Govindan。2016。互联网范围内的流量监管分析。载于 2016 ACM SIGCOMM 会议论文集。468–482页。[10] Sadjad Fouladi、John Emmons、Emre Orbay、Catherine Wu、Riad S Wahby 和 Keith Winstein。2018。Salsify：通过更紧密地整合视频编解码器与传输协议实现低延迟网络视频。载于第15届USENIX网络系统设计与实现研讨会（NSDI 18）。267–282页。[11] Sangtae Ha、Injong Rhee 和 Lisong Xu。2008。CUBIC：一种新型 TCP 友好的高速 TCP 变体。《ACM SIGOPS《操作系统评论》》42卷5期（2008），64–74页。[12] Debajyoti Halder、Prashant Kumar、Saksham Bhushan 和 Anand M Baswade。2021。FybrrStream：一个基于 WebRTC、高效且可扩展的点对点直播平台。载于 2021年国际计算机通信与网络会议（ICCCN）。IEEE，1–9页。[13] Xiangjie Huang、Jiayang Xu、Haiping Wang、Hebin Yu、Sandesh Dhawaskar Sathyanarayana、Shu Shi 和 Zili Meng。2025。ACE：面向高质量实时通信的发送突发性控制。载于 ACM SIGCOMM 2025 会议论文集。1182–1198页。[14] Raj Jain 和 Shawn Routhier。1986。分组列车——测量方法与计算机网络流量新模型。《IEEE通信领域精选期刊》4卷6期（1986），986–995页。[15] Zhidong Jia、Yihang Zhang、Qingyang Li 和 Xinggong Zhang。 [无日期]。通过帧突发拥塞控制应对 RTC 的码率变化。（[无日期]）。[16] Zhidong Jia、Yihang Zhang、Qingyang Li 和 Xinggong Zhang。2024。BurstRTC：通过帧突发拥塞控制利用 RTC 的可变码率。载于第八届亚太网络研讨会论文集。213–214页。[17] Alan B Johnston 和 Daniel C Burnett。2012。WebRTC：HTML5 实时网络的 API 和 RTCWEB 协议。Digital Codex LLC。[18] Srinivasan Keshav。1991。流量控制的控制理论方法。载于通信架构与协议会议论文集。3–15页。[19] Zili Meng、Yaning Guo、Chen Sun、Bo Wang、Justine Sherry、Hongqiang Harry Liu 和 Mingwei Xu。2022。通过最短控制环为无线实时通信实现稳定的低延迟。载于 ACM SIGCOMM 2022 会议论文集。193–206页。[20] Zili Meng、Xiao Kong、Jing Chen、Bo Wang、Mingwei Xu、Rui Han、Honghao Liu、Venkat Arun、Hongxin Hu 和 Xue Wei。2024。Hairpin：重新思考基于边缘的交互式视频流媒体中的丢包恢复。载于第21届USENIX网络系统设计与实现研讨会（NSDI 24）。907–926页。[21] Zili Meng、Tingfeng Wang、Yixin Shen、Bo Wang、Mingwei Xu、Rui Han、Honghao Liu、Venkat Arun、Hongxin Hu 和 Xue Wei。2023。通过自适应帧率实现高质量实时通信。载于第20届USENIX网络系统设计与实现研讨会（NSDI 23）。1429–1450页。[22] Arvind Narayanan、Eman Ramadan、Rishabh Mehta、Xinyue Hu、Qingxu Liu、Rostand AK Fezeu、Udhaya Kumar Dayalan、Saurabh Verma、Peiqi Ji、Tao Li 等。2020。Lumos5G：绘制并预测商用毫米波 5G 吞吐量。载于 ACM 互联网测量会议论文集。176–193页。[23] Craig Partridge、Dennis Rockwell、Mark Allman、Rajesh Krishnan 和 James Sterbenz。2002。让 TCP 更快地启动。BBN Technologies，马萨诸塞州剑桥，BBN 技术报告 8339（2002）。

[24] Devdeep Ray、Connor Smith、Teng Wei、David Chu 和 Srinivasan Seshan。2022。SQP：低延迟交互式视频流媒体的拥塞控制。*arXiv*预印本 *arXiv:2207.11857*（2022）。[25] Grand View Research。2023。《亚太视频流媒体市场规模、份额与趋势分析报告》，按组件（硬件、软件、服务）、解决方案（互联网协议电视、OTT）、流媒体类型（直播/线性、视频点播）、平台（智能手机和平板电脑、智能电视、台式机和笔记本电脑、游戏主机）、服务（咨询、托管服务、培训与支持）、收入模式（订阅、按交易付费、广告、混合）、最终用途（消费者、企业）和国家划分，并提供细分市场预测，2023—2030年。https://www.grandviewresearch.com/industry-analysis/asia-pacific-video-streaming-market 访问日期：YYYY-MM-DD。[26] Sandvine。2024。《2024年全球互联网现象报告》。https://www.sandvine.com/global-internet-phenomena-report-2024。[27] Iván Santos-González、Alexandra Rivero-García、Tomás González-Barroso、Jez-abel Molina-Gil 和 Pino Caballero-Gil。2016。实时流媒体：RTSP 与 WebRTC 的比较研究。载于国际普适计算与环境智能会议。施普林格，313–325。[28] Bruce Spang、Shravya Kunamalla、Renata Teixeira、Te-Yuan Huang、Grenville Armitage、Ramesh Johari 和 Nick McKeown。2023。Sammy：平滑视频流量，成为友好的互联网邻居。载于*2023*年*ACM SIGCOMM*会议论文集。754–768。[29] Farzad Tashtarian、Abdelhak Bentaleb、Hadi Amirpour、Sergey Gorinsky、Junchen Jiang、Hermann Hellwagner 和 Christian Timmerer。2024。{ARTEMIS}：直播视频流媒体的自适应码率阶梯优化。载于第*21*届*USENIX*网络系统设计与实现研讨会（*NSDI 24*）。591–611。[30] Truong Cong Thang、Hung T Le、Anh T Pham 和 Yong Man Ro。2014。HTTP直播流媒体比特率自适应方法评估。*IEEE*《通信领域精选期刊》32，4（2014），693–705。[31] Vinod M Vokkarane、Jason P Jue 和 Sriranjani Sitaraman。2002。突发分段：一种减少光突发交换网络中分组丢失的方法。载于*2002*年*IEEE*国际通信会议。会议论文集。*ICC 2002*（目录号：*02CH37333*），第5卷。IEEE，2673–2677。[32] Shibo Wang、Shusen Yang、Xiao Kong、Chenglei Wu、Longwei Jiang、Chenren Xu、Cong Zhao、Xuesong Yang、Jianjun Xiao、Xin Liu 等。2024。Pudica：面向云游戏拥塞控制中的{近零} 排队延迟。载于第*21*届*USENIX*网络系统设计与实现研讨会（*NSDI 24*）。113–129。[33] David Wei、Pei Cao、Steven Low 和 Caltech EAS。2006。重新审视 TCP 节奏控制。载于*IEEE INFOCOM*论文集，第2卷。Citeseer，3。[34] Francis Y Yan、Hudson Ayers、Chenzhi Zhu、Sadjad Fouladi、James Hong、Keyi Zhang、Philip Levis 和 Keith Winstein。2020。原位学习：一项视频流媒体随机实验。载于第*17*届*USENIX*网络系统设计与实现研讨会（*NSDI 20*）。495–511。[35] Wei Zhang、Tong Meng、Xianhua Zeng、Wei Yang、Changqing Yan、Chao Li、Chengguang Li、Feng Qian、Junfeng Yang、Lei Zhang 和 Zhi Wang。2024。利用 WebRTC 实现大规模直播流媒体。载于*2024*年*ACM SIGCOMM*会议论文集。[36] Xu Zhang、Yiyang Ou、Siddhartha Sen 和 Junchen Jiang。2021。{SENSEI}：使视频流媒体质量与动态用户敏感度相匹配。载于第*18*届*USENIX*网络系统设计与实现研讨会（*NSDI 21*）。303–320。[37] Anfu Zhou、Huanhuan Zhang、Guangyuan Su、Leilei Wu、Ruoxuan Ma、Zhen Meng、Xinyu Zhang、Xiufeng Xie、Huadong Ma 和 Xiaojiang Chen。2019。学习协调移动视频电话中的视频编解码器与传输协议。载于第*25*届移动计算与网络年度国际会议。1–16。[38] Yuhan Zhou、Tingfeng Wang、Liying Wang、Nian Wen、Rui Han、Jing Wang、Chenglei Wu、Jiafeng Chen、Longwei Jiang、Shibo Wang 等。2024。{AUGUR}：面向{实时}流媒体低尾延迟的实用移动多路径传输服务。载于第*21*届*USENIX*网络系统设计与实现研讨会（*NSDI 24*）。1901–1916。

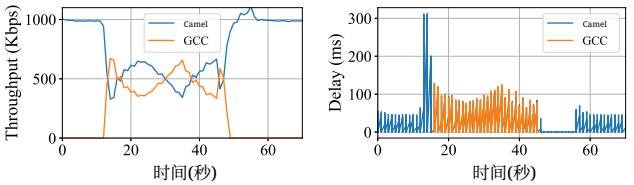
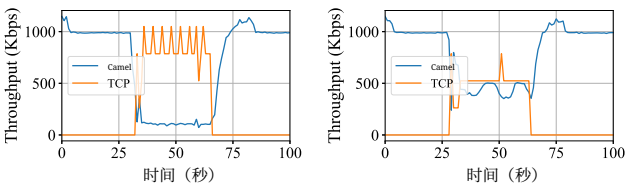


图16：与 GCC 共存。

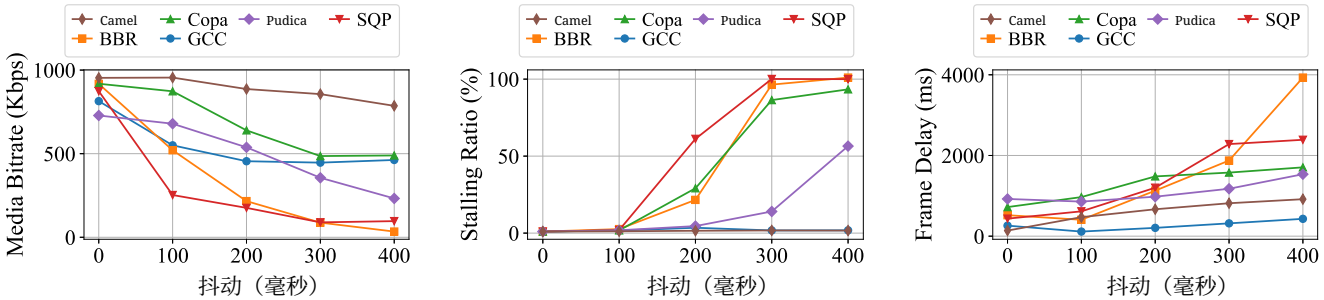


(a) $buffer = 500KB$。 (b) $buffer = 100KB$。

图17：与 TCP 共存。

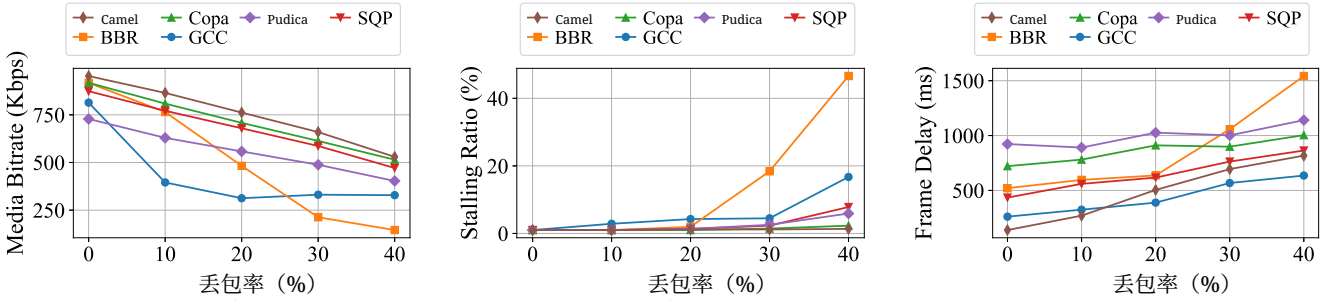
表1：真实世界 LTE网络轨迹上的体验质量比较。

	4G_步行_50167			4G_驾车_50187			5G_步行_90			5G_步行_109		
	比特率（千比特每秒）	延迟 (ms)	卡顿 (%)	比特率（千比特每秒）	延迟 (ms)	卡顿 (%)	比特率（千比特每秒）	延迟 (ms)	卡顿 (%)	比特率（千比特每秒）	延迟 (ms)	卡顿 (%)
Camel	1606.61	1014.57	6.35	2010.09	183.69	0.30	2030.23	135.34	1.42	1981.05	546.51	4.99
Copa	1697.66	997.89	6.78	2035.74	268.16	0.00	2042.70	185.39	1.36	2052.34	652.03	5.05
BBR	1668.18	1167.29	10.92	2014.49	195.75	0.29	2028.53	213.41	1.34	2042.48	687.53	4.84
GCC	1054.97	704.40	6.05	1624.72	139.36	1.02	2037.92	153.65	1.41	1958.70	606.71	5.40
SQP	1419.80	914.53	9.72	1954.26	188.76	2.41	1979.38	187.53	2.67	1860.21	634.29	8.61
Pudica	1535.89	1037.90	7.73	1930.93	275.97	1.20	1946.38	231.36	3.65	1884.94	603.07	9.03
Salsify	1504.33	/	12.69	2041.93	/	2.03	2059.46	/	1.56	2025.78	/	9.59



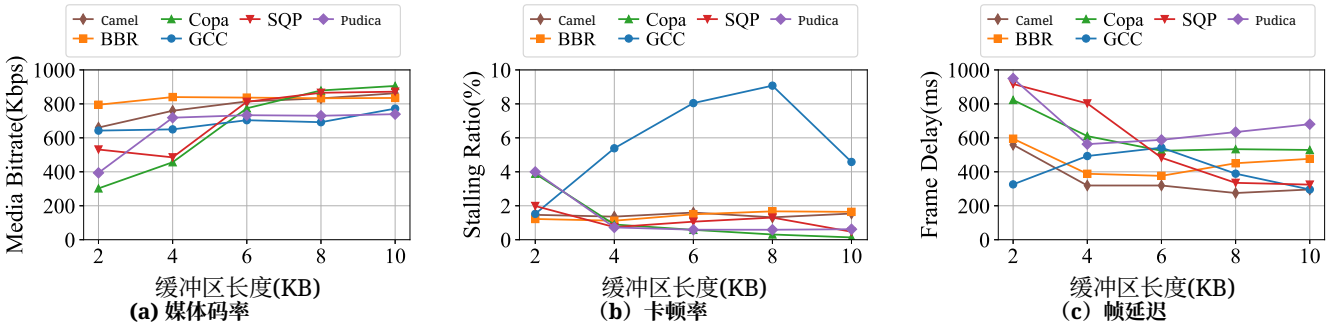
(a) 媒体码率。 (b) 卡顿率。 (c) 帧延迟。

图18：抖动情况下的性能。



(a) 媒体码率。 (b) 卡顿率。 (c) 帧延迟。

图19：丢包率下的性能。



(a) 媒体码率 （b）卡顿率 （c）帧延迟

图20：Camel 在浅缓冲区条件下能够保持高比特率、低卡顿率和低帧延迟 s.