USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Medley: Optimizing Midgress Bandwidth for Commercial Live Streaming CDNs

Haiping Wang, *Fudan University and ByteDance;* Wanxin Shi, *Fudan University;* Sandesh Dhawaskar Sathyanarayana, Shu Shi, and Feng Qian, *ByteDance;* Yinghao Yang and La Zuo, *Fudan University and ByteDance;* Hebin Yu and Ruixiao Zhang, *ByteDance;* Ruoshi Sun, *Fudan University;* Yajie Peng, Xiaofei Pang, Ruili Fang, Zhenpeng Zhu, and Weiqi Chen, *ByteDance;* Yang Xu, *Fudan University*

https://www.usenix.org/conference/nsdi26/presentation/wang-haiping

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

Medley: Optimizing Midgress Bandwidth for Commercial Live Streaming CDNs

Haiping Wang[1,2], *Wanxin Shi*[1], *Sandesh Dhawaskar Sathyanarayana*[2], *Shu Shi*[2]*, *Feng Qian*[2], *Yinghao Yang*[1,2], *La Zuo*[1,2], *Hebin Yu*[2], *Ruixiao Zhang*[2], *Ruoshi Sun*[1], *Yajie Peng*[2], *Xiaofei Pang*[2], *Ruili Fang*[2], *Zhenpeng Zhu*[2], *Weiqi Chen*[2], *Yang Xu*[1]*

[1] *Fudan University* [2] *ByteDance*

Abstract

The rapid expansion of live streaming services in recent years has introduced new cost challenges for Content Delivery Networks (CDNs). Unlike conventional Video on Demand (VoD) streaming, where CDNs can utilize the off-peak hours to preload video content on edge nodes, live streaming requires real-time delivery across CDN nodes before reaching end users, significantly increasing midgress bandwidth costs for service providers. Characteristics such as real-time content generation and consumption, strict requirements for service quality, and systems that need to scale with diverse resources to support the increase in large-scale users, make it difficult to optimize live-streaming midgress traffic with traditional DNS-based and redirection-based methods.

At ByteDance, we have successfully implemented a substream-based CDN architecture to minimize midgress bandwidth for our commercial live streaming services. By dividing a video stream into multiple sub-streams and assigning each to a different edge node, the system can substantially reduce midgress traffic while adapting to the increasing scale of the system and live-stream users. We present Medley, a production-grade substream delivery system, and validate its efficacy through a large-scale, real-world deployment. Running on one of the largest live streaming platforms, Medley handled up to 0.85 million concurrent viewing requests per minute during peak hours. Our evaluations show that Medley reduces midgress costs by 76.83%, while maintaining consistent Quality of Experience (QoE).

1 Introduction

At ByteDance, we operate one of the world's largest live streaming (LVS) systems , where the escalating financial burden of content delivery has become a critical challenge. Our LVS system follows a typical hierarchical CDN design [17, 18, 23, 35], as illustrated in Fig. 1. The overall bandwidth cost consists of two components: *egress* traffic, which flows from L2 node clusters to viewers and grows roughly linearly with the audience size, and *midgress* traffic, which flows from source data centers to L1 node clusters and then from L1 to L2. Although decades of live streaming research have pioneered architectures such as overlay networks [3, 5, 39] and multipath sub-streams [16] to enhance performance and robustness, the challenge of minimizing midgress costs within modern CDNs remains significantly under-explored. In LVS, each live viewer requires a real-time streaming connection from L2 to L1 and the source, contributing 41.56% of total egress traffic during peak hours in our production systems.

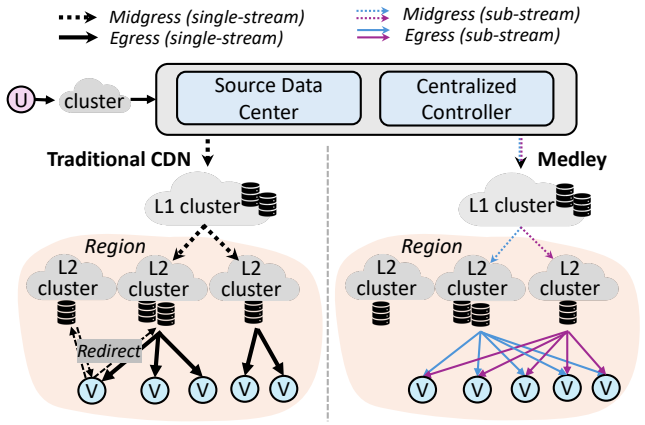


Figure 1: Traditional CDN with a single stream vs. Medley, which splits the same stream into multiple sub-streams (two sub-streams in this figure).

Reducing midgress costs in commercial LVS is inherently difficult since user requests are highly dispersed. Conventional CDNs rely on DNS-based load balancing [24, 26] to spread traffic across a large number of L2 nodes, each requiring its own copy of streams even serving only a few viewers. To mitigate this, we have adopted several optimizations. First, servers within the same L1/L2 cluster share streams via free intra-cluster traffic, avoiding redundant upstream requests. However, this approach doesn't extend across numerous geographically distributed clusters. Second, we deploy a centralized controller similar to VDN [23], to redirect viewers of the same stream to fewer L2 nodes [19,38] (as shown in the left of Fig. 1). In practice, however, this approach is constrained by the limited capacity of L2 resources and the massive volume of stream requests during peak periods. Moreover, the rapid growth of LVS has led us to integrate smaller-capacity edge resources with lower bandwidth cost [36, 37], which further exacerbates the midgress problem.

This paper introduces a "split-and-merge" sub-stream ap-

*Corresponding authors: Yang Xu (xuy@fudan.edu.cn) and Shu Shi (shishu.1513@bytedance.com)

proach to reduce midgress traffic at scale. As shown in the right of Fig. 1, each video stream is split into multiple sub-streams. A viewer reconstructs the original video by connecting to multiple L2 nodes, retrieving all sub-streams, and merging them locally. This design significantly lowers midgress costs: each L2 node only delivers a sub-stream instead of replicating the whole stream. In general, splitting a stream into K sub-streams can theoretically reduce midgress traffic by a factor of up to $\frac{1}{K}$. Conceptually, this is equivalent to combining K nodes into a virtual cluster, enabling them to collectively serve all viewers up to capacity while requiring only a single upstream copy.

However, despite the idea of sub-stream has been extensively utilized in delivery systems for decades [9, 16, 28], how to utilize sub-stream to optimize midgress traffic is non-trivial. Prior works on sub-stream splitting are mainly designed for transmission acceleration, such as aggregating bandwidth and reducing latency via multipath routing [27, 30, 40], recovering packet loss through joint multipath sub-streaming and network coding [16, 21], and tolerating single-point failures and packet losses with erasure coding [16, 28, 41]. In contrast, leveraging sub-stream for midgress optimization in a large-scale commercial LVS system presents several unique challenges. First, how should a sub-streaming system be designed to optimally split and elastically schedule live video, ensuring it can scale to massive real-world traffic while fully realizing its cost-saving potential? Second, multi-node delivery is more vulnerable to performance degradation: the delay or failure of any sub-stream disrupts the entire video. Ensuring that sub-streaming preserves the same Quality of Experience (QoE) as traditional delivery is critical for commercial viability (see §2.3 for details).

We present Medley, the first production-grade substream-based live video delivery system optimized for midgress bandwidth. Medley is fully integrated into ByteDance's large-scale video service (LVS) platform. It incorporates three fundamental system designs to enable sub-stream delivery and reduce midgress costs without compromising QoE.

Sub-stream Splitting: Choosing the sub-stream number of K in the large-scale system needs to be careful (§3.1). Although splitting the video stream into K sub-streams can theoretically reduce midgress traffic by up to a factor of $1/K$, however, it will also proportionally increase the connection overhead and system unreliability. This raises several questions. First, how should K be chosen to optimally balance bandwidth cost and system overhead? Second, how can K be made elastic to adapt to stream popularity change and CDN load? To address these problems, we model the selection of K as an optimization problem that minimizes midgress cost, system overhead, and unreliability. On top of the model, we propose a sub-stream scalable algorithm to make elastic sub-streams scheduling decisions to well adapt to live streaming dynamics(§4.2).

Sub-stream Transmission: Splitting live streams into multiple sub-streams can introduce two forms of blocking at receiver-side transmission (§3.2): *intra-sub-stream blocking*, where a missing or delayed frame blocks data transfer within its own sub-stream; and *inter-sub-stream blocking*, where delays in one sub-stream propagate to block others. These blocking issues increase the risks of rebuffering and first-frame delays. To address this, we implement a buffer-driven fast frame requests that proactively retrieve missing frames in the application buffer while suppressing false recovery triggers to reduce redundancy. This ensures multi-node streaming achieves the same smoothness as single-stream (§4.3.2).

Sub-stream Generation: When generating sub-streams content, a naive allocation of video frames to sub-streams might cause the traffic and frame rate imbalance, given with different frame types (P/I/B)(§3.3). To avoid imbalances that reduce efficiency, we propose a hybrid Weighted Round-Robin (WRR) strategy which considers frame type and size variation to ensure a balanced sub-stream generation(§4.3.1). We also integrate the sub-stream generation workflow into CDN delivery pipelines with minimal change to existing CDN services.

Medley has been deployed at scale in ByteDance's live streaming system, serving up to 0.85 million concurrent viewers per minute. Large-scale A/B testing shows that Medley reduces midgress bandwidth cost by 76.83% for sub-streaming traffic without degrading user QoE. While Medley introduces an additional average end-to-end (E2E) delay of 0.155s compared to a single-stream CDN baseline (i.e., redirection), our A/B tests confirm that this increase has a negligible statistical impact on QoE, given that the expected E2E constraint in commercial live streams is approximately ∼2-3 seconds [4, 29].

In summary, we make the following contributions:

- Based on our analysis of midgress bandwidth cost in live production, we argue for a fundamental shift in addressing midgress cost at scale. To this end, we present Medley, the first streaming system that splits streams for midgress optimization and demonstrate its benefits in a large-scale production deployment serving millions of users (§2).
- Medley proposes three fundamental designs for its realization: choosing the optimal sub-stream number K, content-aware sub-stream generation that avoids imbalance, and a buffer-based solution to mitigate intra- and inter-sub-stream blocking without excessive false recovery (§3 & §4).
- A large-scale evaluation of Medley, including deployment experience, shows that it reduces midgress bandwidth cost by 76.83% without degrading viewer QoE (§5).

This work raises no ethical concerns; all data are anonymous and do not contain personally identifiable information.

2 Background

2.1 ByteDance's Live Streaming CDN

CDN Architecture. Our live streaming service is built on a hierarchical CDN architecture designed for scalability and cost efficiency [15, 43]. As shown in Fig. 1, the uplink begins when a user publishes a video to a nearby cluster, which relays the stream to the *source data center*. At the source, the

video is processed for distribution, including transcoding into multiple bitrates for adaptive bitrate (ABR) streaming.

The distribution network typically follows a tree structure with multiple layers. At the leaf level are *L2 node*s (also known as edge cluster [23]), which are geographically distributed and by default assigned via DNS to serve users in their respective regions. If an L2 node cluster does not have the requested live stream, it forwards the request to an upper-level *L1 node* (also known as reflector [23]), which in turn retrieves the video from the source data center on a cache miss. L1 clusters thus act as intermediaries between the source and the numerous L2 clusters, preventing tens of thousands of L2s from directly querying the source to ensure scalability.

The system also implements a centralized scheduling controller with global network visibility, similar to [18,23]. While requests are initially resolved by local DNS to the nearest L2 node, the controller may redirect traffic to other nodes (even geographically farther ones) before the session starts, for optimizing service quality and resource utilization, e.g., local nodes are overloaded.

CDN System	**Overall MER**
LVS w/ DNS	0.416
LVS w/ Redirect	0.236

Table 1: The MER in the commercial CDN system

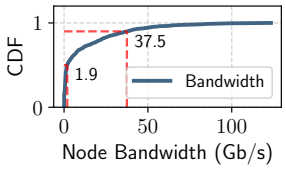


Figure 2: The bandwidth CDF of L2 node clusters

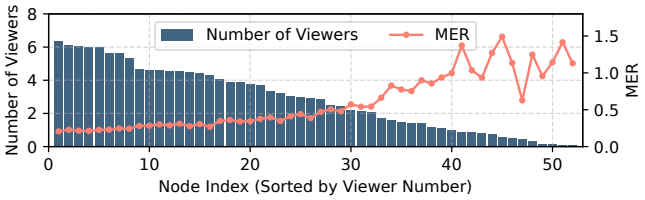


Figure 3: The number of viewers & MER on each L2 node of stream with popularity of 130 during peak time 21:00-21:05

Midgress Cost Problem. Bandwidth cost constitutes roughly 90% of CDN operational expenses, spanning both *egress* (traffic from L2 nodes to end users) and *midgress* (traffic between the source, L1, and L2 nodes). As noted in prior work [31], midgress traffic constitutes a substantial bandwidth overhead in conventional VoD CDNs. This problem becomes even more severe in live streaming scenarios, where each L2 node is required to subscribe to streams from upstream L1 nodes and the source in real time. Unlike in VoD systems, this overhead cannot be effectively mitigated by prediction or pre-caching.

To quantify this, we define the Midgress–Egress Ratio (MER) as the ratio of midgress to egress bandwidth. MER measures the internal efficiency of a CDN system (or a single cluster): a lower MER indicates that the midgress traffic is shared across more viewers, achieving lower cost. Table 1 summarizes the overall MER of our production LVS CDN systems. The system using only DNS-based load balancing achieves an MER of 0.416, nearly half the total bandwidth, incurring tens of millions of dollars in annual costs.

The elevated LVS midgress cost arises from scattered user requests and the requirement for real-time delivery. Fig. 3 presents a snippet from our production dataset to illustrate this dispersed request pattern: an unpopular stream, which has ∼130 concurrent viewers, is distributed via the default DNS policy to nearby nodes based on user location. Even for this relatively small stream, the system requires 52 nodes to replicate upstream traffic, with nearly half of these nodes serving fewer than two users*. This dispersion results in redundant midgress traffic and an inflated MER.

2.2 Limitations of Existing Optimizations

Existing Cost Saving Methods. A variety of approaches have been proposed to lower LVS cost, but few directly address midgress traffic. Techniques such as super-resolution [14], novel codecs [10], and ABR [13,32], aim to reduce the bitrate of each stream, thereby lowering both egress and midgress traffic. Other solutions, such as offloading traffic to cheaper PCDNs [36], mainly reduce egress cost. While effective in their respective domains, these methods do not directly reduce the MER and are largely orthogonal to our approach.

In practice, several mechanisms are commonly employed to reduce midgress traffic, and we have deployed them in our production system for cost efficiency. First, intra-cluster sharing ensures the video stream can be shared via free intra-cluster traffic, which is typically excluded from midgress cost. Therefore, for each video stream, a cluster as a whole only sends one request to the upper-layer for subscription. However, this optimization cannot extend across clusters, since inter-cluster traffic is billed as midgress. Second, L1 and L2 nodes automatically stop retrieving streams from upper layers once there are no active subscribers, avoiding unnecessary traffic. Most importantly, *redirection* [19,38] has proven to be the most effective method: with assistance from the centralized controller, users are redirected from local DNS-assigned nodes (often "cold" nodes) to alternative clusters (often "hot" nodes) already serving the requested stream. This aggregation reduces midgress traffic because fewer L2 nodes subscribe to streams from upper L1. As shown in Table 1, enabling redirection lowers the overall MER from 0.416 to 0.236.

Limitations of Redirection. Ideally, redirecting all the viewers of the same stream to a single L2 node would minimize midgress cost. In practice, however, this is infeasible in a system serving millions of streams to hundreds of millions of viewers. In our production, redirection successfully applies to only 61% of requests. The primary bottleneck is edge cluster capacity and highly concurrent stream requests. Over years of

*Notably, some L2 nodes exhibit an MER exceeding 1, as they temporarily retain streams from upper-layer nodes following the exit of all subscribers.

operation, our LVS infrastructure has evolved to add a large number of medium-sized edge clusters to offload traffic from large-capacity data centers to support the rapid growth of traffic scale. Since cluster bandwidth costs are billed at peak usage and do not scale linearly, it is more cost-efficient and quality-friendly to distribute traffic across multiple medium-sized clusters closer to users than serve them with a single large data center[†]. Fig. 2 shows the CDF of bandwidth across L2 clusters: the median is only 1.9 Gbps—insufficient to serve a medium-popular online stream with 3000 viewers. In practice, most L2 clusters are already heavily utilized during peak hours, serving hundreds of concurrent streams, and lack spare capacity to absorb redirected user requests at scale.

Another limitation is that only new sessions can be redirected. Ongoing sessions cannot be moved without disrupting playback, which would significantly degrade QoE. Consequently, nodes serving only a small number of users cannot be reclaimed until all sessions naturally end, delaying reallocation and reducing efficiency. These limitations underscore the need for new approaches that reduce midgress traffic more effectively, while making full use of existing infrastructure.

2.3 From Single-stream to Sub-stream

Bring Substream into CDN. We propose a novel approach to optimize the midgress cost. Instead of delivering a complete video stream, the system divides the video into multiple parts (i.e., sub-streams), which are distributed across different L2 nodes. At the client side, these partial components are aggregated to reconstruct the original stream. To illustrate the potential gains, consider a video stream with bitrate F served to 5 viewers, where ach L2 node has only $3F$ bandwidth available (Fig. 1). Using the traditional single-stream delivery, it requires at least two L2 nodes to accommodate all 5 viewers, generating a total midgress traffic of $2F$, and an MER of 0.4. In contrast, the sub-stream method splits the video into two sub-streams, with each L2 subscribing to only one sub-stream ($\frac{1}{2}F$). All 5 viewers need to connect to both L2 nodes and retrieve all sub-streams to restore the original video, resulting in total midgress traffic of F, and an MER of 0.2—a 50% reduction compared to the single-stream solution. Note that in this case, the MER reduction is achieved without using additional capacity as each L2 node only needs $\frac{5}{2}F$ to deliver sub-streams to all 5 viewers.

Theoretically, we model the midgress cost of a substream-based CDN and show that splitting a stream into k substreams can reduce midgress traffic by a factor of $1/k$ for the same egress volume. We provide the formal proof in Appendix A.1.

Limitations of Existing Splitting Schemes. While the theoretical benefits of substreams are compelling, realizing them in a production live-streaming environment is non-trivial. Prior research on substream-based transmission primarily focuses on two paradigms: *Network Coding(NC)* and *Multipath*.

- *Network Coding Systems* [16, 21, 34] employ erasure coding to partition a live stream into k data substreams and m parity substreams as redundant protection. This approach ensures robust delivery where the receiver can reconstruct the original stream by fetching any k out of the $k+m$ substreams, providing resilience against unpredictable network losses.
- *Multipath Systems* [3, 16, 27, 33, 40] schedule video packets across multiple paths to aggregate bandwidth and reduce transmission latency. These systems often rely on content redundancy, such as packet duplication [3, 16, 42] and reinjection [40], to mitigate tail latencies and improve QoE.

Despite their effectiveness in reliability and speed, existing frameworks are ill-suited for midgress traffic management due to two fundamental limitations: First, *existing schemes are inherently redundancy-oriented.* NC-based and multipath-based approaches intentionally inflate traffic to combat packet loss or jitter. For instance, network coding increases the total egress traffic by a factor of $\frac{m}{k}$ for encoding packets. In a CDN environment, this overhead typically outweighs the bandwidth savings gained from midgress reduction, rendering such methods impractical for large-scale deployment. For midgress optimization, a redundancy-free splitting strategy is essential. Second, *existing mechanisms are performance-oriented rather than cost-oriented.* In prior work, the number of substreams k is often a static parameter dictated by the available physical paths. However, as our analysis in §A.1 demonstrates, k is a critical decision variable that directly governs the midgress-to-egress ratio (MER) and must be dynamically optimized. Furthermore, a practical splitting strategy must jointly consider the complexities of content distribution and global load balancing across edge nodes. In summary, leveraging substreams for midgress optimization in large-scale CDNs remains an unexplored and unvalidated design space.

Medley's Challenges. To our knowledge, Medley is the first system to introduce substreams into a large-scale CDN specifically for midgress optimization. Unlike prior work, Medley employs a redundancy-free splitting strategy to maximize cost savings without incurring egress overhead. To ensure reliability without parity packets, we leverage extended server-side caching and proactive upstream re-fetching to handle losses. Scaling this mechanism to millions of concurrent users presents two fundamental challenges: **(C1)** determining the optimal splitting number of sub-streams K to trade off cost and overhead, and elastically scheduling them across CDN nodes with varying load demands; and **(C2)** designing stream transmission solutions that mitigate both intra-sub-stream and inter-sub-stream blocking, avoiding rebuffer and ensuring QoE. We detail the design principles of Medley in §3.

3 Design Pillars of Medley

3.1 The Implication of Splitting $\mathbb{K}$ Sub-streams

Splitting live streams into K sub-streams seems simple in concept but is complex in practice.

[†]Constructing large-scale data centers demands ISPs deploy dedicated network routes and entails substantial operational costs, while small-scale clusters can acquire third-party servers and utilize existing infrastructure.

Tradeoff between Cost and System Overhead. Consider a live streaming scenario as depicted in Fig. 4, where a 2Mbps stream is watched by 400 viewers. Each L2 node has 200Mbps of bandwidth to serve this stream, meaning that four such nodes are required to serve all users. While simplified, this example clarifies key concepts. With a single stream (K = 1), midgress traffic is 2 × 4 = 8Mbps. Splitting into K = 2 sub-streams reduces midgress traffic to 1 × 4 = 4Mbps, a 50% reduction, and increasing to K = 4 further reduces traffic to 0.5 × 4 = 2Mbps, yielding a 75% gain. This raises the question: can K be increased indefinitely?

While larger K values potentially reduce midgress costs, they also increase system overhead equally. For example, with K = 2, each L2 node must serve 200 user connections instead of 100, and each viewer must connect to two nodes instead of one to reconstruct the full stream. This doubles the connection overhead and introduces a critical reliability concern: if any L2 node fails, the stream cannot be reconstructed on the user device. Hence, the question becomes *how to select K to balance bandwidth cost, system overhead, and reliability?*

Embracing the Elasticity for K. During live streaming, the subscriber counts of streams can fluctuate dynamically, and the optimal K changes accordingly. When this occurs, the centralized scheduler should reassign the number of L2 nodes allocated to the sub-streams and adjust the splitted sub-stream content on nodes. This adjustment can result in streaming failures and user interruptions.

Consider the case in Fig. 4(c) where the viewer population drops from 400 to 200, indicating K changes from 4 to 2 to minimize splitting overhead since two nodes are enough to handle such traffic. This also prompts the scheduler to reduce the number of active L2 nodes from four to two. However, the scheduler is unaware that serving states of users, simply drops the nodes from four to two and informs L1 nodes to redistribute content to 2-sub-streams and flush L2 node cache. These changes result in connection failures for the sub-stream system. In a single-stream system, users can seamlessly continue downloading without being affected by content flushing since the replicated content on nodes is identical. In contrast, with sub-streams, users previously connected to four L2 nodes suddenly lose access to data from two of them, resulting in incomplete stream construction. Similarly, when the nodes are increased from two to four, i.e., the value of K increases, viewers need to connect to two additional nodes to receive a complete stream, instead of the previous two nodes. These dynamics introduce significant complexity in maintaining seamless streaming with sub-streams. This raises the second design choice, *how to make K elastic?*

Medley's design. The optimal tradeoff between cost and overhead can be achieved by designing a utility model that jointly captures both factors to determine the optimal number of sub-streams, K. In Section §4.2, we detail how this model is constructed and optimized. To further support elasticity, we adopt a control-plane/data-plane decoupled design: the control plane initially partitions streams into a fixed number of sub-streams and dynamically adjusts them based on node availability and stream popularity, while the data plane simply forwards sub-streams following the scheduling decisions.

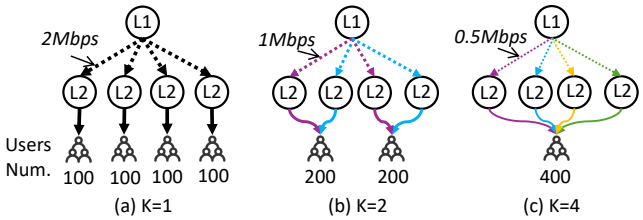


Figure 4: Increasing # of sub-streams (K) from 2 to 4 reduces cost from 4 Mbps to 2 Mbps (50% gain).

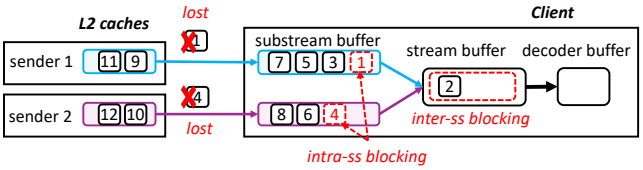


Figure 5: Intra and inter-blocking with sub-streams

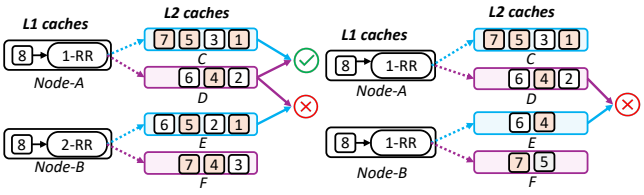
3.2 Stream Buffer to Solve intra-inter Blocking

Once we move from single-stream to sub-streams, an important question arises: how do we transmit with multiple sub-streams and handle sub-stream blocking without affecting user QoE? Since sub-streams are delivered by different L2 nodes over separate connections to users, a missing frame can cause blocking within its own sub-stream (intra-sub-stream blocking) or across different sub-streams (inter-sub-stream blocking). This problem is analogous to Head-of-Line (HoL) blocking observed in multipath transport [9, 30].

As illustrated in Fig. 5, consider a two-sub-stream system where Frame 1 is lost. This loss first blocks the forwarding of subsequent frames of sub-stream 1 into the stream buffer, a phenomenon we refer to as intra-sub-stream blocking. Furthermore, the missing Frame 1 blocks the transfer of frames of sub-stream 2, such as Frame 2, from the stream buffer to the decode buffer due to decoding dependency, resulting in a rebuffer. We refer to this as inter-sub-stream blocking. A similar blocking occurs with the loss of Frame 4, which prevents Frame 6 and Frame 8 from being pushed forward. As the number of sub-streams *K* increases beyond two, the probability of inter-sub-stream blocking rises.

Although Frame 1 can be recovered in sub-stream buffer and eventually pushed into the stream buffer, the issue remains unresolved if Frame 4 is still missing, as it continues to block progress from the stream buffer. This scenario increases recovery latency, requiring simultaneous recovery across multiple sub-streams. What's more, if Frame 1 and Frame 4 belong to the first GOP, the blockage will also affect the first-frame latency, which is easily noticed by users and degrades QoE.

Medley's design. Whether intra-stream or inter-stream blocking, the stream buffer—being the last hop before the application layer—provides the most accurate vantage point. It



(a) Independent local scheduler (b) Asynchronous caching

Figure 6: The generation of sub-streams (1-RR/2-RR denotes round-robin allocation of every/every two frames)

can directly measure blocking, detect missing frames, and trigger recovery. To leverage this, we adopt a unified solution that pushes sub-stream frames in the same way as a single stream, and employ a fast frame-request algorithm to avoid these blocking issues discussed above(§4.3.2).

3.3 Content-aware Sub-stream Generation

Finally, we need to generate sub-stream content to enable sub-stream-based CDN delivery. This process entails determining how to allocate video frames to multiple sub-streams and where to generate them.

Consistent Stream Construction. Suppose we implement the sub-stream generation at L1 nodes, which distributedly split sub-streams to L2 nodes. Consider that local scheduling is performed independently at Node A& B (L1 node) as illustrated in Fig. 6a. Nodes A and B implement a round-robin scheduler, but do so without awareness of the other node's assignments, which can lead to inconsistent sub-stream generation results and incomplete streams at the client. Node A assigns Frame 3 to Node C, while Node B assigns Frame 3 to Node F. When the user downloading the stream connects to Nodes D and E, and fails to reconstruct the stream without Frame 3. A similar issue arises when L1 nodes join the live stream at different times. As shown in Fig. 6b, if Node A joins the stream at time T_i (starting from key Frame 1) and Node B joins at T_{i+1} (starting from key Frame 4), users connected to Node D and E cannot reconstruct the complete stream.

Content-aware Frame Allocation. A further challenge arises when the schedulers allocate sub-streams without considering frame size variability. In live streaming, I-frames can be an order of magnitude larger than P-frames, and two orders larger than B-frames. The sub-stream allocation cannot break the load balance of the CDN. In the example of Fig. 6a, where orange blocks represent I frames and others are P frames, Node A overloads Node C by allocating too many I frames, while Node D remains underloaded—an inefficiency that does not occur in single-stream scenarios.

Medley's Design. Instead of generating sub-streams in a distributed manner at L1 nodes, we implement a universal sub-stream generation mechanism. This ensures consistent sub-stream generation and integrates seamlessly into the CDN delivery pipeline. Additionally, a content-aware scheduler can better capture frame variations and produce a more balanced allocation of video frames (detailed in §4.3.1).

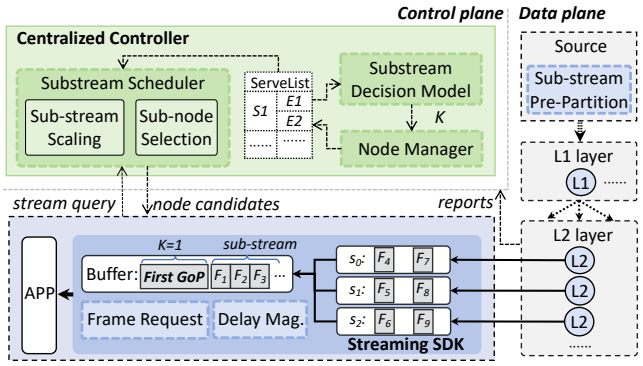


Figure 7: Medley System

4 Medley as a System

4.1 System Overview

Medley has three core components: (1) sub-stream number and scheduling decision, (2) sub-stream content generation, and (3) multi-node transmission. Since Medley is designed to operate analogously to traditional single-stream live streaming, its components are integrated into both the **control plane** (centralized controller) and the **data plane** (including CDN node delivery and client-side mechanisms in the downlink).

Fig. 7 illustrates how these components are realized within the Medley architecture. The control-plane logic begins with the centralized controller, which invokes the sub-stream decision model to determine the number of sub-streams (K). Based on this decision, the controller uses the sub-stream scaling module to adjust the sub-stream number and uses the subnode selection module to allocate the appropriate set of subnodes. These nodes are returned to users to subscribe and indicate L1 nodes to distribute the original stream across the selected subnodes to form K sub-streams.

On the uplink path, when a publisher starts a live stream, the stream is first pushed to the source data center. The incoming stream is processed by the frame partition module and annotated with sub-stream metadata, enabling it to be split independently across nodes. Specifically, the L1 nodes perform sub-streams separation by respectively forwarding them to the designated L2 nodes chosen by the control plane. Node management maintains all real-time telemetry and user information, which the controller uses for subnode selection and dynamic adjustment during live streaming.

On the downlink path, when a client subscribes to a stream, it first sends a request to the central controller, which returns the list of nodes selected for all sub-streams of that stream. The client then fetches the sub-streams from these nodes. The client's stream management module retrieves these sub-streams in parallel, manages both intra-stream and inter-stream blocking, and reassembles them into a single, synchronized playback stream. By embedding Medley's logic into both the CDN infrastructure and the client, the system enables coordinated, end-to-end sub-stream management. This tight integration ensures that splitting, scheduling, and playback are fully aligned, enabling high-quality live streaming

while optimizing midgress cost.

In Medley, the system is hybrid with single-stream and sub-stream. The subscription between the L1 layer and the source data center still uses single-stream, as the MER of this layer is already low. We enable sub-streams for L1-to-L2 delivery to reduce L2 layer midgress. Additionally, streams determined by the model to use K=1 are equally to use single-stream - these are typically cold streams that already achieve optimal midgress with the existing method. Medley also streams the first GOP via single-stream to ensure fast startup and serve as a backup for sub-stream delivery. The centralized controller carefully manages them to provide system efficiency.

4.2 The Scheduling Controller

Here, we describe how Medley splits sub-streams, by modeling K, making it elastic, and scheduling them across nodes.

4.2.1 Modeling the Optimal $\mathbb{K}$ for Sub-streams

We formalize the problem as the minimization of midgress bandwidth $B(k)$, management overhead $M(k)$, and system unreliability $R(k)$ for a selected k value, which is operationalized through the utility function $U(k)$ as defined in Eq. 1.

$$U(k) = w_1 \cdot B(k) + w_2 \cdot M(k) + w_3 \cdot R(k) \tag{1}$$

The coefficients w_1, w_2 , w_3 allow tuning of the tradeoffs in production, with their implications discussed in the implementation §A.2. The corresponding optimization problem is therefore expressed as a minimization problem $\mathcal{P}1$.

$$\begin{aligned} \mathcal{P}1: \quad & \underset{k}{\text{minimize}} \quad U(k) \\ \text{s.t.} \quad & k \geq 1, \\ \text{where} \quad & B(k) = b \cdot \lceil \frac{vv}{\lfloor \frac{B}{n \cdot \frac{b}{k}} \rfloor} \rceil \\ & M(k) = vv \cdot k \\ & R(k) = 1 - d^k \end{aligned}$$

Here, B denotes the bandwidth of L2 nodes, and n denotes the number of streams that a node can serve. b represents the bitrate of the stream with vv viewers.

- *Midgress cost* $B(k)$. The midgress cost can be calculated by the traffic of each sub-stream ($\frac{b}{k}$) multiplying serving nodes number($\lceil \frac{vv}{\lfloor \frac{B}{n \cdot \frac{b}{k}} \rfloor} \rceil$) as well as sub-streams number(k). The curve is a monotonically non-increasing function of k.
- *Management overhead* $M(k)$. After splitting sub-streams, the number of connections in the system will be k times of the viewers count (vv), which grows linearly with K.
- *System Unreliability* $R(k)$. Suppose the probability of a single connection operating normally is d, then the probability that all connections operate normally is d^k. The unreliability probability introduced by sub-streams is calculated as $1 - d^k$.

In production, the model takes real-time system metrics from the monitoring module as input and makes a K-value decision at the stream level. This K-value result is then used to guide node selection for sub-stream scheduler and client transmission. To ensure stable system operation, we update the model output at a 5-minute granularity. Popularity and other input metrics are calculated based on the average of statistics collected within each 5-minute interval.

4.2.2 The Elastic $\mathbb{K}$ for Sub-streams

As discussed in our design section §3.1, stream popularity in live streaming systems is dynamic, and this dynamism leads to varying K values optimized by the sub-stream model. To support sub-streams across a variable number of nodes, we design a sub-stream pre-partitioning and dynamic merging mechanism, which decouples K from the dynamics of user subscription status and ensures its elasticity.

Elastic K via Pre-Partition and L1 Forwarding. We use a constant m to pre-partition streams into sub-stream units $\{1, 2, \ldots, m\}$, which is defined as the basic unit of sub-stream generation. These m sub-stream units are non-overlapping yet collectively form the original video stream. This enables adjusting the number of sub-streams to any K by reorganization of units, eliminating the need for repartitioning and flushing L2 node cache states. L2 nodes are configured as simple forwarding units, with the controller making assignment decisions and communicating them to viewers via downlink. As the number of sub-streams and L2 nodes changes, new viewers receive updated node lists and K values, while existing users can continue viewing without interruption. This ensures connectivity continuity during K adjustments.

Achieving Elasticity with Merge. As shown in Algorithm 1, we use a simple yet effective merging algorithm to combine m pre-partitioned units into K sub-streams, where the target $K \leq m$. Let S_i denote the list of unit IDs (sids) belonging to merged sub-stream i where $0 \leq i < K$. Given the target K, the algorithm calculates how many sids to group into each sub-stream (Lines 3-7). The units of merged sub-streams are treated as a single sub-stream: they are assigned to the same node (instead of different nodes) for users and subscribed to via a single connection (instead of multiple). This scaling method enables dynamic adjustment of sub-streams number – eliminating the need to re-partition sub-streams in the data plane while remaining compatible with both existing requests (using the old K) and new requests (using the updated K).

4.2.3 Node Selection for $\mathbb{K}$ Sub-streams

Once the K decision is made and optimized, Medley needs to select K nodes from the CDN node pool. Node selection is a prerequisite for solving K or making it elastic and follows a set of rules. The scheduler first prioritizes nodes already in the sub-stream's service list to maximize utilization of current midgress nodes. When these service-list nodes are full, the scheduler selects additional nodes from the pool using two simple criteria for rapid decision-making: (1) nodes in the same region and ISP, to preserve the time sensitivity of live streaming; and (2) nodes with sufficient bandwidth, exceeding the required capacity, to allow room for expansion.

To ensure reliability within sub-streams, each selected node is provisioned with multiple backup server IPs. If a primary node IP or connection fails, the client can fetch sub-streams

from these backup IPs. They are also used by clients to recover missing frames. The backup IPs correspond to different machines within the same L2 node cluster, enabling clients to switch without incurring additional midgress traffic and while preserving time sensitivity. Finally, the information about all allocated nodes for the sub-streams is compacted into a single response packet sent to the user.

4.3 The Data Transmission

In this section, we first introduce how Medley generate sub-stream content, and then discuss client-side transmission handling with intra-sub-stream and inter-sub-stream blocking.

4.3.1 Content Aware Sub-stream Generation

Content-aware WRR. We propose a scheduler integrated with Weighted Round Robin (WRR), leveraging a key insight: frame size variations exist but are relatively stable within each type—all I-frames, for example, have similar sizes, as do P-frames and B-frames. We construct separate sub-stream queues for each frame type and apply independent WRR scheduling across sub-streams for each type. Same-type frames are assigned to sub-streams to balance traffic (ensured by WRR), while per-type independent allocation balances the frame rate. This design, termed the Content-aware Weighted Round Robin (CWRR) scheduler, is detailed in Algorithm 2.

Lightweight Sub-stream Generation. We implement a lightweight sub-stream generation mechanism to ensure generation consistency. The CWRR scheduler is integrated into the source datacenter, which assigns frames into sub-streams as required (i.e., m pre-partitioned sub-stream units). Each frame is marked in its metadata with *sid* indicating which sub-stream it belongs to—this tag stays the same no matter how the frame is distributed within the system. Next, tagged frames of all sub-streams are sent to L1 nodes, which simply check the sid tags and forward the frames to the downstream L2 nodes that have subscribed to the corresponding sub-stream. L2 nodes are only responsible for sub-stream-level subscriptions for users. This simple yet universal mechanism allows sub-streams to be consistently delivered in different nodes within the CDN system, ensuring that any L2 nodes requesting sub-streams will receive identical content.

Algorithm 1: Sub-stream Scaling Algorithm

Input: The runtime sub-stream decision number K, pre-partitioned number m
Output: The sub-stream merge result S

// Dynamic Scaling: Maintain K substream sets S_i, each initialized as an empty set. The algorithm uniformly aggregates m partition units into these K substream sets.
$S = \Phi$, $sid = 0$
for *each i in [0, K-1]* **do**
  for *each j in [1,* $\frac{m}{K}$*]* **do**
    $S_i \leftarrow sid$++
    if $sid >= m$ **then**
      break
  end
end

Algorithm 2: Frame Scheduler Algorithm

Input: Incoming frame F with size of m
Output: Decide the *sid* tag for F

// Initialization: Maintain three per-frame-type queues for each substream i. Let Q_i^j be the queue $j \in \{I,P,B\}$ of substream i, and w_i^j be its traffic weight, calculated as the cumulative size of all frames in the queue.
for *each i in [0, k]* **do**
  for *each j in [I, P, B]* **do**
    $Q_i^j = \Phi$, $w_i^j = 0$;
  end
end
// Per-frame Allocation: assign the incoming frame F to the substream *sid* with the minimum traffic load.
$j' \leftarrow typeof(F)$, $sid \leftarrow min\{w_i^j\}, i \in [0,k]$;
$Q_{sid}^{j'} = Q_{sid}^{j'} \cup F$;
$w_{sid}^{j'} = w_{sid}^{j'} + m$;

4.3.2 Multi-Node Sub-streams Transmission

The client employs a multi-node transmission framework to subscribe to sub-streams from different nodes in parallel and merge them to reconstruct the original video. Given that the first-frame delay is perceptible to users, Medley uses the single-stream to accelerate first-GOP delivery. It also designates single-stream as a fallback for sub-stream transmission failures. To mitigate sub-stream blocking in a dynamic network, we introduce two additional key optimizations.

Mitigation of Intra and Inter Sub-stream Blocking. To mitigate intra-sub-stream blocking, we implement a UDP-based unreliable transport that delivers data to the application layer at the frame level rather than as a continuous stream. We avoid QUIC or TCP due to their inherent reliability mechanisms, which are unnecessary for our design. Once the sub-stream transport receives all packets of a completed frame, it directly moves the frame from the sub-stream buffer to the application buffer, delegating frame recovery to the application layer.

At the stream buffer in the application layer, we employ a fast frame request algorithm that proactively requests missing frames in the buffer to prevent rebuffering. The algorithm introduces a *sub-stream out-of-order degree* metric, d_{ooo}, which quantifies the severity of frame blockage by measuring the gap between the maximum received frame number and the largest consecutive frame in the buffer. A high d_{ooo} indicates persistent gaps that require urgent recovery. Backup nodes are leveraged for rapid frame recovery.

In the stream buffer, the client calculates two parameters: ooo_{max} (Eq. 2) and d_{ooo}, representing the maximum tolerable

out-of-order delay before rebuffering and the degree of disorder, respectively. ooo_{max} is calculated as the remaining buffer length, which equals the maximum buffer threshold (T_{buffer} minus the current end-to-end delay and retransmission time (Eq. 2). It triggers an immediate retransmission request for the first blocking frame if d_{ooo} exceeds ooo_{max}. The request is routed to the sub-stream's backup node, avoiding the potentially congested primary node.

$$ooo_{max} = (T_{buffer} - e2e - RTT) \cdot FPS \quad (2)$$

Turning off the False Triggers. If some sub-streams are always left behind others, we might see that the inter-sub-stream blocking always exists and clients generate a huge amount of frame recovery in order to avoid rebuffering. Note that this case of blocking is not caused by lost frames in some sub-stream; it is caused by the late arrival of sub-stream where the delay difference of the fastest sub-stream with the late sub-stream exceeds the buffer length threshold T_{buffer}. This causes unnecessary waste of data. Hence, we also need to manage the delay of sub-stream to let the delay difference of sub-stream be less than the buffer threshold. The client will periodically monitor the sub-stream delay difference and proactively switch the sub-stream to backup nodes if the delay is significantly larger than the fastest sub-stream.

5 In-Production Evaluation

The Medley system has been deployed in the large-scale production environment of ByteDance. The total bandwidth capacity of CDN nodes in Medley is approximately 15 Tbps. These nodes are widely distributed to serve all viewers in China. Since March 2023, Medley has begun serving real-world users. Now, its scale has expanded to handle an average user traffic of 10 Tbps at peak time, where the number of concurrent viewing users reaches 0.85 million.

We evaluate Medley from two aspects: i.e., cost reduction and user QoE. Specifically, we compare Medley with traditional CDN through real-world A/B experiments. For traditional CDN, we consider two widely-used baselines: CDN-DNS and CDN-redirection. CDN-DNS primarily relies on DNS to allocate users to the nearest node cluster; while CDN-redirection, which is also the online version of our commercial system, whenever new requests arrive, the centralized scheduling controller determines whether requests need to be migrated to another node for aggregation (instead of following DNS recommendations). All baselines incorporate practical optimizations that deliver top-tier live streaming experiences to billions of users worldwide. The three solutions are compared through rigorous A/B tests adhering to industry standards. Each testing cycle involves tens of millions of real-world users daily and typically runs for several weeks until results stabilize. To ensure experimental fairness, we fairly select user traffic such that live streams assigned to each testing group exhibit similar popularity and viewer distributions.

5.1 Midgress Cost Reduction

We use the *MER* metric to evaluate how much the midgress cost can be reduced by Medley. The MER metric is calculated as midgress bandwidth divided by egress bandwidth.

Overall Midgress Cost. Fig. 8 illustrates the average peak-hour (21:00–22:00) MER of the entire system across the three solutions over a full week. Note that while the actual A/B test spanned several weeks, we selected one week of data here to demonstrate how this metric fluctuates over time. The average MER values for Medley, CDN-redirection, and CDN-DNS are 0.0547, 0.2361, and 0.4156, respectively. This indicates that for the same number of viewers (i.e., identical egress traffic), Medley reduces midgress costs by 76.83% compared to CDN-redirection (the state-of-the-art midgress reduction solution) and by 86.83% compared to CDN-DNS.

Node-level Midgress Cost. We further computed the node-level MER to assess the per-stream unit midgress cost at individual nodes. This metric is calculated as the ratio of a stream's midgress bandwidth consumption to the egress bandwidth at a node. Figure 10 illustrates the CDF of node-level MER for Medley (employing sub-streams) and the CDN-redirection baseline (utilizing single-streams). The results reveal a substantial reduction in unit midgress cost with Medley: the 50th-percentile MER for the single-stream CDN stands at 0.240, while it decreases to 0.049 for the sub-stream-based Medley. This confirms that the reduction in midgress achieved by adopting sub-streams stems from its ability to lower the per-stream unit midgress cost at nodes effectively.

Deep Dive into Medley's Substream Splitting. We further assess the efficiency of sub-stream splitting in Medley system.

- *Sub-stream scaling.* Figure 11 shows the distribution of sub-stream counts in Medley. Approximately 68% of streams use 5 sub-streams, while the remaining 32% are dynamically scaled down to 1–4 by the controller as their popularity decreases to varying extents. This result confirms that Medley's scalability effectively adapts with popularity dynamics to minimize the overhead of splitting.
- *Sub-stream splitting.* We also evaluated whether Medley balances split live streams into sub-streams by measuring two metrics: the frame max-min ratio and the traffic max-min ratio. These ratios are calculated as the average frame rate/traffic of the largest sub-stream divided by that of the smallest sub-stream. In production, the average traffic max-min ratio is 1.11 and the frame max-min ratio is 1.03, confirming that sub-streams are split in a balanced way.

5.2 User QoE

We conducted A/B experiments and long-term user engagement experiments to validate that Medley has no impact on user QoE compared to single-stream-based CDN baselines.

Streaming Quality. As shown in Fig. 9, we use four metrics to evaluate the streaming quality of Medley and baselines. *1) First-Frame Delay.* The average first-frame delays of Medley and CDN-DNS are nearly identical (315ms vs 316ms),

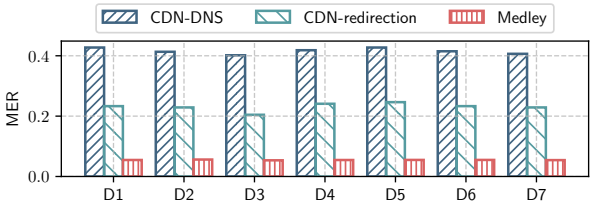


Figure 8: The Midgress Cost of Medley and CDN baselines

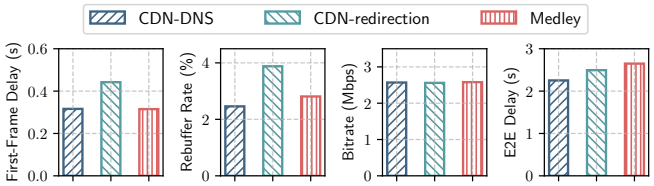


Figure 9: The viewing QoE of Medley and CDN baselines

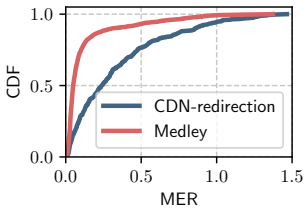


Figure 10: The CDF of Node-level MER of Medley and CDN

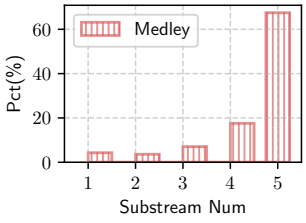


Figure 11: The distribution of sub-stream number in Medley

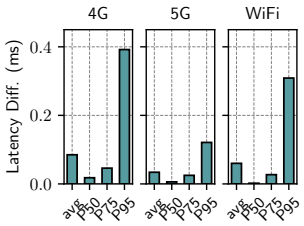


Figure 12: The delay heterogeneity of sub-streams

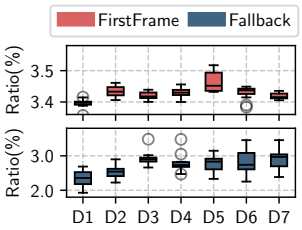


Figure 13: The ratio of first GOP and fallback traffic

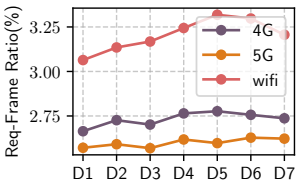


Figure 14: The traffic ratio of fast requested frames in different network types

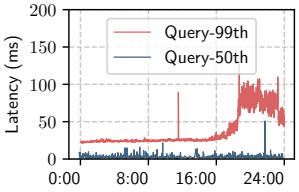


Figure 15: The 50th and 99th latency of requests query in the centralized scheduler

as Medley leverages single-stream for first-GOP delivery. In contrast, CDN-redirection exhibits a significantly higher delay of 442ms due to the additional network hop introduced by redirection. *2) Rebuffer Rate.* The rebuffer rate of Medley is 2.81%, slightly higher but comparable to CDN-DNS with 2.46%. The CDN-redirection, however, experiences a much higher rebuffer rate of 3.88%, 1.5 times that of CDN-DNS. This is because redirection aggregates traffic to specific nodes, and given the highly dynamic nature of traffic patterns, these nodes are prone to overload, leading to quality degradation. *3) Bitrate.* There is no significant difference in bitrate across all solutions. *4) E2E Delay.* Medley increase E2E delay by 397ms compared to CDN-DNS, and increased by 155ms compared to CDN-redirection. The additional delay in Medley arises from the stream buffer implemented on user devices to mitigate sub-stream blocking. This delay is tightly controlled, ensuring the E2E delay remains within our product's 3-second requirement. Notably, media delivery and interactive messages utilize separate pipelines: sub-streams are exclusively applied to non-interactive media delivery, making minor E2E delay increases imperceptible to users [‡]. While Medley increases latency by 0.155s compared to the redirection baseline—i.e., the existing solution running in production, the substantial cost benefits it delivers render this trade-off fully acceptable for a super-large-scale production system. Furthermore, we also conduct long-term experiments to confirm that Medley exerts no negative impact on user engagement and business metrics (discussed in the following part).

User QoE. To assess whether Medley affects user experience—particularly whether the slight increase in end-to-end delay influences user engagement—we conducted a long-term observational experiment comparing Medley with the single-stream CDN system (i.e., CDN-DNS) to evaluate user QoE. A six-month A/B test was conducted, with each experimental group accounting for 860,000 daily views. We employed two key industry-standard metrics to measure user QoE: user viewing duration, which reflects users' willingness to continue watching the live stream, and purchase proportion, which indicates the proportion of users making purchases during their viewing session. Table 2 presents the long-term observation results. The average user duration differed by -0.42%, and the proportion of users making purchases differed by +0.05%. All these metric variations fall within the range of normal fluctuations with very small p-values. These results confirm that Medley maintains a user experience comparable to that of the single-stream CDN system.

Deep Dive into Medley's Delivery. We further assess the transmission efficiency of sub-stream delivery in Medley.

- *Substream delay heterogeneity.* Fig. 12 presents the distribution of sub-stream latency differences across network types. Averages are minimal: 60ms (WiFi), 34ms (5G), 85ms (4G). Even at the 95th percentile, differences remain insignificant

[‡]Interactive messages (e.g., comments, purchases) continue to use traditional protocols and remain unaffected by Medley.

(121–392ms). These are far smaller than multi-path delay difference [40], as we connect nodes within the same ISP and network interface, avoiding cross-ISP/interface issues.

• *First-GOP and fallback traffic.* Fig. 13 presents the daily distribution of traffic ratios for first GOP and fallback traffic fetched via the single-stream mechanism, with respective averages of 3.41% and 2.72%. This indicates that the majority of traffic is handled by sub-streams.

• *Fast Frame request.* Fig. 14 shows application-level frame request traffic ratio: 4G (2.73%), 5G (2.59%), Wi-Fi (3.20%). This discrepancy reflects Wi-Fi's greater loss proneness than cellular networks. We also observed 1.34% average data redundancy in client buffers, likely from network dynamics causing substream RTT variations and false loss detections—delayed (not lost) packets arriving post-request introduce redundancy. Overall, redundancy is acceptable with optimization potential in the future.

5.3 System Performance

We validate that the system performance of Medley meets the required standards with an acceptable increase in connections and resources with the adoption of sub-streams.

Connection Number Increase. As shown in Tab. 3, compared to single-stream CDN, the average number of connections processed per node in Medley increased from 1.125k to 4.19k (increased by 272%). This is because Medley uses multiple connections to support sub-streams delivery. The average CPU rate of nodes also increased by 1.97% in Medley in order to handle more connections. However, the average egress bandwidth per node of Medley is still comparable to single-stream CDN. This shows that the increased number of connections doesn't affect the node serving performance.

User Device Energy Consumption. We also evaluated the performance overhead on user devices for transmitting with multiple connections. We measured the CPU utilization and device temperature. As shown in Table 3, compared to the single-stream CDN, Medley exhibited increases of 0.73% in CPU utilization and 2.83% in temperature, respectively. These increases are insignificant and acceptable to users.

Playing Failure Rate. We use the playing failure rate to evaluate the system's unreliability, which is calculated as the proportion of user sessions that fail to see the playing of pictures and exit the live stream session. As shown in Table 3, the failure rate of Medley is +0.80% compared to traditional CDN. This increase is not significant, and validate Medley still meets the system reliability standard.

Controller Performance at Scale. To evaluate the response latency of the centralized scheduler after integration with Medley's logic, we measure the 50th and 99th percentile query latency, which is defined as the time from receiving the request to the result returned at the controller. These 99th latency metrics assess worst-case response latency. As shown in Fig. 15, the 50th latency of query requests remains around 2ms in both peak and non-peak periods. The 99th latency is 23ms during non-peak times but rises to 70–90ms at peak times. This increase stems from prolonged node selection under higher QPS, particularly since Medley must select nodes for K sub-streams. However, the response time still remains acceptable for second-level live streaming scenarios.

Medley *v.s.* CDN	User Duration	Purchase Proportion
Value Diff(%)/P-value	-0.42/0.30	+0.05/0.90

Table 2: The Key User Engagement Metrics in Business

Medley v.s. CDN	Node Conn.	Node CPU	Egress Traffic	User Device CPU	User Device Tempe.	Fail Rate
Diff(%)	+272	+1.97	+0.07	+0.73	+2.83	+0.80

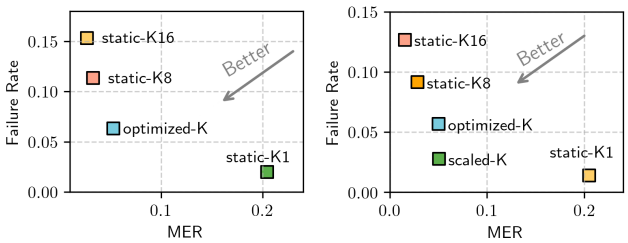
Table 3: System Performance Metrics and Failure Rate

6 In-Controlled Study of Medley

To validate the optimization efficacy of Medley's design, we constructed a Mininet-based testbed to facilitate controlled experiments, which cannot be conducted in online environments, as untested designs may degrade user experience. This testbed integrates all core components of Medley, including concurrent clients, CDN nodes, and a centralized scheduling controller, enabling the emulation of a small-scale live streaming system operating with sub-streams, supporting tens of nodes and hundreds of clients.

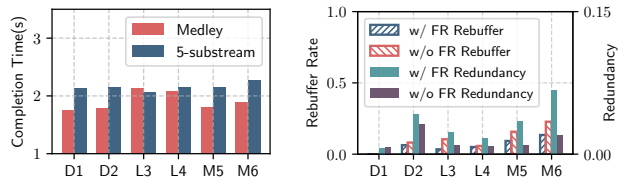
The Optimization of Sub-stream Number Decision. To evaluate whether Medley's decision-making of sub-streams number K optimizes cost and performance, we compare Medley's decision model and scaling mechanisms with several straightforward strategies to decide K: 1) *static-K1* method, which uses the traditional single-stream method; 2) *static-K8* which uses a fixed value of 8 sub-streams, a value determined based on experience, with no adjustments for changes during service; 3) *statuc-K16* which are same with static-K8 but use 16-sub-streams; 4) *optimized-K* which adopts a fixed K, where this value is derived by Medley's model using the peak popularity of the stream; 5) *elastic-K* which dynamically adjusts K through Medley scaling method, where K is determined by the model based on the stream's real-time popularity. We ran with two groups of settings of viewing patterns: stable popularity where the viewership of streams remains the same during experiments, and varying popularity where the viewership of streams changes over time. We compare the overall MER and system overhead of all strategies. The default crash rate of a network link is set to 0.01.

• *Setting-1: Stable popularity.* We pushed four live streams into the system, with each stream configured to tens to hundreds of viewers and run for 30 minutes. The number of concurrent viewers remained static throughout the experiment. Fig. 16a presents the overall MER versus failure rate under different strategies. Since failure rate and connection overhead are similarly affected by the parameter K, we only plot the failure rate in this figure for clearer illustration—note that both the *optimized-K* and *scaled-K* strategies still in-



(a) Stable popularity setting (b) Varying popularity setting

Figure 16: The trade-off between cost and overhead under different popularity settings



(a) The first-GOP completion time via single-stream v.s. multiple sub-streams

(b) The rebuffer rate and redundancy rate with and without fast frame request (FR)

Figure 17: Transmission performance in dynamic network

corporate connection overhead into their decision-making processes. Evidently, the *optimized-K* strategy achieves the optimal trade-off between midgress cost and overhead. In contrast, *static-K1*(i.e., single-stream) incurs the highest midgress cost; *static-K8* and *static-K16* adopt excessively large numbers of sub-streams, leading to high overhead.

• *Setting-2: Varying popularity.* Building on the above setup, we further introduced variability to the concurrent viewer count of live streams: half of the clients exited every 10 minutes during streaming. Fig. 16b presents the test results under this dynamic popularity pattern. Clearly, *scaled-K* outperforms *optimized-K* and all static strategies, as it adjusts the number of sub-streams to minimize utility cost in line with popularity dynamics—this adaptability allows Medley to adapt with system changes effectively.

Transmission Performance in Dynamic Network. We validated the necessity of Medley's transmission designs: first-GOP acceleration and fast frame request. We used the TC tool [12] to simulate different weak network conditions: D1 and D2 represent fluctuated RTTs scenarios; L3 and L4 represent lossy links scenarios; and M5 and M6 represent mixed loss and delay scenarios. First, we assessed the first-GOP completion time using single-stream and 5-sub-stream transmission under different network conditions. As shown in Fig. 17a, downloading the first GOP via sub-streams results in longer completion times than using a single stream(1912ms vs. 2156ms on average)—especially under heterogeneous delay conditions. This confirms the need for Medley's design of transmitting the first GOP via a single stream.

Next, we measured the rebuffer rate and redundancy rate with and without Medley's fast frame request algorithm (denoted as FR) under different weak networks. As shown in Figure 17b, enabling fast frame requests significantly reduces the average rebuffer rate, effectively handling network dynamics. We also confirmed that increased redundancy typically occurs with dynamic sub-stream delays: D1 and D2 show a much higher redundancy rate compared to pure lossy weak network conditions (L3 and L4).

7 Related Work

Cost Reduction. While egress optimization is well-studied, midgress efficiency remains largely overlooked. Bitrate-reduction techniques like super-resolution [11, 20] and advanced codecs [6, 10] are orthogonal to midgress efficiency, as they reduce total volume without improving the *MER*. Offloading frameworks [36, 37] shift traffic to cheaper resources but cannot reduce midgress demand. ABR schemes [13, 32] may even counter-intuitively exacerbate midgress costs. By forcing L2 nodes to fetch multiple bitrate variants of the same stream, these methods fragment popularity and degrade *MER*.

Redirection-based Optimization. Request redirection is a standard technique to reduce midgress costs by leveraging existing content at forwarding nodes. While modern CDNs [7, 8, 19, 22] and major cloud providers [2] offer redirection to adapt to dynamic traffic, this approach often incurs significant QoE penalties. Furthermore, its effectiveness remains constrained by edge bandwidth volatility in large-scale live streaming environments.

Substream-based Transmission Framework. Multipath transport [3, 27, 33, 40] and network coding-based approaches [16, 21, 34] leverage similar substream concepts, focusing on optimizing packet scheduling and data encoding to maximize delivery performance and reliability. In contrast, Medley employs substreams to minimize midgress overhead within CDNs, prioritizing cost-oriented splitting and the global-scale orchestration of substreams.

8 Conclusion

In this paper, we presented Medley, a new design for reducing midgress cost in large-scale LVS systems. To the best of our knowledge, Medley is the first production system to demonstrate the effectiveness of stream splitting to reduce midgress bandwidth at scale. Our deployment results show that sub-streaming can substantially lower bandwidth costs without compromising QoE, making it a promising approach for the next generation of commercial LVS platforms.

9 Acknowledgements

We are grateful to the NSDI reviewers for their constructive critique, and to our shepherd Ramesh Sitaraman in particular, for his valuable comments, all of which have helped us greatly improve this paper. This work is sponsored by the National Key Research and Development Program of China (No. 2024YFE0203900), National Natural Science Foundation of China (U25B2032, 62572128, 62402118), and Shanghai Pujiang Project Funding (24PJD003).

References

[1] Video File Format Specification Version 1.0. https://rtmp.veriskope.com/pdf/video_file_format_spec_v10.pdf, 2012.

[2] Configure URL Redirection. https://www.alibabacloud.com/help/en/cdn/user-guide/configure-url-redirection, 2024.

[3] Konstantin Andreev, Bruce M. Maggs, Adam Meyerson, and Ramesh K. Sitaraman. Designing overlay multicast networks for streaming. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, page 149–158, New York, NY, USA, 2003. Association for Computing Machinery.

[4] Abdelhak Bentaleb, May Lim, Mehmet N Akcay, Ali C Begen, Sarra Hammoudi, and Roger Zimmermann. Toward one-second latency: Evolution of live media streaming. *IEEE Communications Surveys & Tutorials*, 2025.

[5] Suk Kim Chin. An efficient streaming method in overlay multicast networks. In *2009 IEEE 9th Malaysia International Conference on Communications (MICC)*, pages 78–83, 2009.

[6] Mallesham Dasari, Kumara Kahatapitiya, Samir R Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, 2022.

[7] Jie Deng, Gareth Tyson, Felix Cuadrado, and Steve Uhlig. Internet Scale User-Generated Live Video Streaming: The Twitch Case. In *Proceedings of International Conference on Passive and Active Measurement (PAM)*, 2017.

[8] Qilin Fan, Hao Yin, Libo Jiao, Yongqiang Lyu, Haojun Huang, and Xu Zhang. Towards Optimal Request Mapping and Response Routing for Content Delivery Networks. *IEEE Transactions on Services Computing (TSC)*, 14(2):606–613, 2021.

[9] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684, March 2020.

[10] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-Latency Network Video through Tighter Integration Between A Video Codec and A Transport Protocol. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[11] Pan Hu, Rakesh Misra, and Sachin Katti. Dejavu: Enhancing Video Conferencing with Prior Knowledge. In *Proceedings of ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2019.

[12] Bert Hubert et al. Linux advanced routing & traffic control howto. *Netherlabs BV*, 1:99–107, 2002.

[13] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar. Online Learning for Low-latency Adaptive Streaming. In *Proceedings of ACM Multimedia Systems Conference (MMSys)*, 2020.

[14] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-Enhanced Live Streaming: Improving Live Video Ingest Via Online Learning. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2020.

[15] Christian Koch, Johannes Pfannmüller, Amr Rizk, David Hausheer, and Ralf Steinmetz. Category-Aware Hierarchical Caching for Video-On-Demand Content on Youtube. In *Proceedings of ACM Multimedia Systems Conference (MMSys)*, 2018.

[16] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky. A transport layer for live streaming in a content delivery network. *Proceedings of the IEEE*, 92(9):1408–1419, 2004.

[17] Federico Larumbe and Abhishek Mathur. Under the hood: Broadcasting Live Video to Millions. *Facebook Code*, 2015.

[18] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, et al. Livenet: A Low-Latency Video Transport Network for Large-Scale Live Streaming. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2022.

[19] Jiayi Liu, Qinghai Yang, and Gwendal Simon. Congestion Avoidance and Load Balancing in Content Placement and Request Redirection for Mobile CDN. *IEEE/ACM Transactions on Networking (TON)*, 26(2):851–863, 2018.

[20] Zhenxiao Luo, Zelong Wang, Miao Hu, Yipeng Zhou, and Di Wu. LiveSR: Enabling Universal HD Live Video Streaming with Crowdsourced Online Learning. *IEEE Transactions on Multimedia (TMM)*, 25:2788–2798, 2022.

[21] Enrico Magli, Mea Wang, Pascal Frossard, and Athina Markopoulou. Network coding meets multimedia: A review. *Trans. Multi.*, 15(5):1195–1212, August 2013.

[22] Ricky K. P. Mok, Vaibhav Bajpai, Amogh Dhamdhere, and K. C. Claffy. Revealing the Load-Balancing Behavior of YouTube Traffic on Interdomain Links. In *Proceedings of International Conference on Passive and Active Measurement (PAM)*, 2018.

[23] Matthew K Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, Real-Time Centralized Control for CDN-Based Live Video Delivery. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[24] Jianping Pan, Y Thomas Hou, and Bo Li. An overview of dns-based server selections in content distribution networks. *Computer Networks*, 43(6):695–711, 2003.

[25] Roger Pantos and William May. HTTP Live Streaming. https://datatracker.ietf.org/doc/html/rfc8216, 2017.

[26] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. Improving content delivery using provider-aided distance information. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 22–34, 2010.

[27] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. *SIGCOMM Comput. Commun. Rev.*, 41(4):266–277, August 2011.

[28] K. V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the facebook warehouse cluster. In *Proceedings of the 5th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'13, page 8, USA, 2013. USENIX Association.

[29] Yongtao Shuai and Thorsten Herfet. Towards reduced latency in adaptive live streaming. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4. IEEE, 2018.

[30] Varun Singh, Saba Ahsan, and Jörg Ott. Mprtp: Multipath considerations for real-time media. In *Proceedings of ACM Multimedia Systems Conference (MMSys)*, 2013.

[31] Aditya Sundarrajan, Mangesh Kasbekar, Ramesh K Sitaraman, and Samta Shukla. Midgress-Aware Traffic Provisioning for Content Delivery. In *Proceedins of USENIX Annual Technical Conference (ATC)*, 2020.

[32] Farzad Tashtarian, Abdelhak Bentaleb, Hadi Amirpour, Sergey Gorinsky, Junchen Jiang, Hermann Hellwagner, and Christian Timmerer. ARTEMIS: Adaptive Bitrate Ladder Optimization for Live Video Streaming. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024.

[33] Haiping Wang, Zhenhua Yu, Ruixiao Zhang, Siping Tao, Hebin Yu, and Shu Shi. Twinstar: A practical multipath transmission framework for ultra-low latency video delivery. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9234–9242, 2023.

[34] Mea Wang and Baochun Li. Network coding in live peer-to-peer streaming. *Trans. Multi.*, 9(8):1554–1567, December 2007.

[35] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 25–34, 2009.

[36] Huanhuan Zhang, Congkai An, Anfu Zhou, Yifan Zhu, Weilin Sun, Yixuan Lu, Jiahao Chen, Liang Liu, Huadong Ma, and Aiguo Fei. Venus: Enhancing QoE of Crowdsourced Live Video Streaming by Exploiting Multiflow Viewer Assistance. In *Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2024.

[37] Rui-Xiao Zhang, Haiping Wang, Shu Shi, Xiaofei Pang, Yajie Peng, Zhichen Xue, and Jiangchuan Liu. Enhancing Resource Management of the World's Largest PCDN System for On-Demand Video Streaming. In *USENIX Annual Technical Conference (ATC)*, 2024.

[38] Rui-Xiao Zhang, Changpeng Yang, Xiaochan Wang, Tianchi Huang, Chenglei Wu, Jiangchuan Liu, and Lifeng Sun. Aggcast: Practical cost-effective scheduling for large-scale cloud-edge crowdsourced live streaming. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 3026–3034, 2022.

[39] Yuchao Zhang, Huahai Zhang, Peizhuang Cong, and Wendong Wang. Rond: Rethinking overlay network design with underlay network awareness. *Proc. ACM Netw.*, 2(CoNEXT2), June 2024.

[40] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, Qing An, Hai Hong, Hongqiang Harry Liu, and Ming Zhang. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2021.

[41] Tianli Zhou and Chao Tian. Fast erasure coding for data storage: A comprehensive study of the acceleration techniques. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 317–329, Boston, MA, February 2019. USENIX Association.

[42] Yuhan Zhou, Tingfeng Wang, Liying Wang, Nian Wen, Rui Han, Jing Wang, Chenglei Wu, Jiafeng Chen, Longwei Jiang, Shibo Wang, Honghao Liu, and Chenren Xu. Augur: practical mobile multipath transport service for low tail latency in real-time streaming. NSDI'24, USA, 2024. USENIX Association.

[43] Behrouz Zolfaghari, Gautam Srivastava, Swapnoneel Roy, Hamid R. Nemati, Fatemeh Afghah, Takeshi Koshiba, Abolfazl Razi, Khodakhast Bibak, Pinaki Mitra, and Brijesh Kumar Rai. Content Delivery Networks: State of the Art, Trends, and Future Roadmap. *ACM Computing Surveys (CUSR)*, 53(2), 2020.

A Appendix

A.1 Single-Stream Cost *vs* Sub-Stream Cost

We use a model to explains how splitting a video stream into k sub-streams, can theoretically reduce MER by a factor of $1/k$, without requiring additional bandwidth resources. Consider b as the bitrate of the video stream and B as the available bandwidth of an edge node. Assuming $B \geq b$, an edge node can serve $\lfloor B/b \rfloor$ users at a given time. If there are n users requesting the video, the system requires at least $\left\lceil \frac{n}{\lfloor B/b \rfloor} \right\rceil$ edge nodes. The egress traffic remains $n \cdot b$, while the midgress traffic is $\left\lceil \frac{n}{\lfloor B/b \rfloor} \right\rceil \cdot b$. The MER of the system can be calculated using Eq. 3.

$$MER_{single\_stream} = \frac{\left\lceil \frac{n}{\lfloor B/b \rfloor} \right\rceil \cdot b}{b \cdot n} \tag{3}$$

When the video stream is divided into k sub-streams, each with a bitrate of b/k, each edge node can serve $\lfloor k \cdot B/b \rfloor$ users. However, since each user must connect to k nodes, the system requires $\left\lceil \frac{k \cdot n}{\lfloor k \cdot B/b \rfloor} \right\rceil$ edge nodes to serve all n users. The MER of such a sub-stream system is calculated as in Eq. 4:

$$MER_{sub\_stream} = \frac{\left\lceil \frac{k \cdot n}{\lfloor k \cdot B/b \rfloor} \right\rceil \cdot \frac{b}{k}}{b \cdot n} \tag{4}$$

We compare $MER_{single\_stream} \cdot k$ with MER_{sub_stream}.

1. $MER_{single\_stream} = \frac{b \cdot \lceil \frac{n}{\lfloor B/b \rfloor} \rceil}{b \cdot n} = \frac{\lceil \frac{n}{\lfloor B/b \rfloor} \rceil}{n}$

2. $k \cdot MER_{sub\_stream} = \frac{b \cdot \lceil \frac{k \cdot n}{\lfloor k \cdot B/b \rfloor} \rceil}{b \cdot n} = \frac{\lceil \frac{k \cdot n}{\lfloor k \cdot B/b \rfloor} \rceil}{n}$

The numerator of MER_{single_stream} is $\lceil \frac{n}{\lfloor B/b \rfloor} \rceil \cdot b$ while that of $k \cdot MER_{sub\_stream}$ is $\lceil \frac{k \cdot n}{\lfloor k \cdot B/b \rfloor} \rceil \cdot b$. The key comparison lies in the denominators inside the ceiling functions, i.e., $\lfloor B/b \rfloor$ and $\lfloor k \cdot B/b \rfloor$. Given the property that $\lfloor k \cdot B/b \rfloor \geq k \cdot \lfloor B/b \rfloor$, it follows that $\frac{k \cdot n}{\lfloor k \cdot B/b \rfloor} \leq \frac{n}{\lfloor B/b \rfloor}$.

Therefore, given that $\lfloor k \cdot B/b \rfloor \geq k \cdot \lfloor B/b \rfloor$, we can conclude:

$$MER_{sub\_stream} \leq \frac{1}{k} \cdot MER_{single\_stream} \tag{5}$$

A.2 Implementation of Medley

Model Parameter Selection. We first introduce coefficient parameter selection for the sub-stream model in § 4.2.1. The three components—midgress cost B, system reliability R, and management overhead M—are first scaled to the same value space to enable additivity. We empirically set the parameters w_1, w_2, and w_3 to 0.7, 0.1, and 0.2, respectively, with midgress assigned the highest weight (as bandwidth costs dominate expenses), followed by system reliability (as it influences the system SLA) and management overhead.

Sub-stream Pre-partition and Buffer Control. In the production, we set the pre-partitioned constant m as 5. This is offline derived by the model using the different popularity sampled from online system statistics as shown in Fig. 18, where the most optimal K values of different popularity are less than or equal to 5. The buffer threshold T_{buffer} at the client is set to 3s since in our system, where we deliver live streaming service with 3-second end-to-end latency.

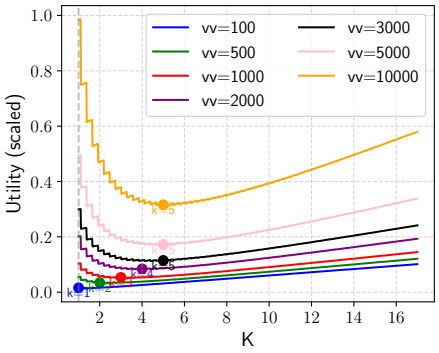


Figure 18: The decision curve of the sub-stream model, with popularity and system statistics sampled from online as input

Integrate with Mobile APPs. The basic protocol stack of Medley is implemented based on UDP. We have implemented unreliable transport and a multi-connection transmission framework. On top of it, we implement the user-level sub-stream recovery strategies. The transport layer SDK is written in C language and can be integrated into different mobile apps (a live stream application of ByteDance). Test packages are released on the client every two weeks, enabling users with updated versions to use the Medley service.

Deploy in Edge Servers. The Medley server is written in C language (implementing UDP-based transmission with Medley client) and deployed on CDN servers with a multi-processor architecture. The function of requesting sub-streams from CDN is implemented based on HTTP-FLV [1, 25] which is compatible with existing CDN architecture.

Transport Protocol and ABR Interaction. Medley utilizes a lightweight UDP-based transport protocol aligned with QUIC. The individual connections between the client and an L2 node employ BBR as the default congestion control algorithm to ensure high throughput and low latency. We do not explicitly optimize for inter-flow competition or fairness, as these challenges are orthogonal to our architecture and have been extensively addressed by prior research on multipath congestion control. The substream mechanism is designed to be transparent to the live ABR (Adaptive Bitrate) logic. In a traditional single-stream deployment, a bitrate switch triggers the client to terminate its current session and initiate a new request for a different Flow ID from an L2 node. Medley extends this workflow to the substream level: a bitrate transition requires the client to concurrently re-subscribe to the corresponding set of substreams for the target bitrate across multiple L2 nodes.