RLive: Robust Delivery System for Scaling Live Streaming Services

Yu Tian[†§], Gerui Lv[†§], Qinghua Wu[†§‡], Ruili Fang[¶], Yajie Peng[¶], Zhichen Xue[¶], Rui Han[¶], Chuanqing Lin[†§], Xiaofei Pang[¶], Ri Lu[¶], Zhenyu Li[†§‡]
[†]Institute of Computing Technology, Chinese Academy of Sciences [¶]ByteDance
[§]University of Chinese Academy of Sciences [‡]Purple Mountain Laboratories

Abstract

As the demand for streaming services surges, content delivery network (CDN) operators face increasing pressure to scale live video delivery without proportionally increasing infrastructure costs. While best-effort edge resources offer a cost-effective extension to traditional CDN capacity, their limited bandwidth and unstable performance pose significant challenges. Our operational experience shows that naively layering such resources onto existing CDN infrastructure falls short in meeting performance and scalability demands. This paper presents RLive, a robust delivery system that scales CDN capacity by integrating best-effort edge resources. RLive features a redundancy-free multi-source data plane to support reliable and cost-efficient live streaming, along with a multi-layer collaborative control plane that combines the global view with local adaptability for scalable user-to-node mapping. Deployed in ByteDance CDN to support large-scale live streaming services with hundreds of millions of daily viewers, RLive has tripled delivery capacity while reducing rebuffering events by 14.9–20.1%.

CCS Concepts: • **Computer systems organization → Distributed architectures**; • **Networks → Network services**; **Network design and planning algorithms**.

Keywords: Content delivery network, Live streaming services, Distributed system, Quality of experience (QoE)

1 Introduction

The explosive growth of crowdsourced live video streaming and e-commerce live streaming has driven an unprecedented surge in the number of live streams and viewers [39, 55, 72, 78, 87]. Content delivery networks (CDNs) have long served as the backbone of live stream delivery [35], where the content is generated in real time and demands stringent latency guarantees. However, CDN operators face growing pressure to scale delivery capacity while controlling operational costs [78, 82]. In response, many operators have begun to leverage underutilized *best-effort edge resources* that are primarily owned and operated by Internet Service Providers (ISPs), such as base stations or apartment gateway devices. Compared to existing dedicated CDN nodes, these resources offer better proximity to end users and are more cost-effective [62, 67, 83].

Yet simply offloading traffic to these edge resources as an extended layer of dedicated CDNs is not enough. A key shift in the live streaming landscape is the rising importance of user-perceived Quality of Experience (QoE), typically measured by end-to-end (E2E) latency, rebuffering ratio, and video bitrate. Even small degradations in QoE can lead to sharp drops in user engagement [17, 40, 88]. This requirement rules out peer-assisted delivery systems that allow end hosts directly serve content to each other for reduced server load, as they were primarily cost-driven and often failed to meet the stringent performance guarantees in modern live streaming services [4, 7, 13, 32]. Essentially, our goal is to leverage the best-effort edge resources for scaling live streaming services, while matching the QoE of dedicated CDN infrastructure.

Our deployment experience underscores the challenges to achieve the above goal. As a major CDN operator supporting multiple live streaming platforms with hundreds of millions of daily active users, we were among the first to explore the use of best-effort nodes for large-scale live streaming delivery. Our initial approach was straightforward. We extended our CDN with only high-availability best-effort nodes–specifically, the top 1% ranked by bandwidth capability and stability. However, this naive attempt resulted in significant increases in both E2E latency (26%—35%) and the rebuffering rate (37.5%—44.7%), indicating that simply integrating the most promising best-effort nodes cannot ensure QoE. In other words, we faced a critical dilemma: *How to fully harness all the best-effort edge resources to scale live streaming services without sacrificing latency or streaming quality?*

We found that even the top 1% of best-effort nodes suffer from low stability and limited capacity, which fundamentally

Co-first authors: Yu Tian, Gerui Lv. Corresponding author: Zhenyu Li.

Conference EuroSys '26, Edinburgh
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

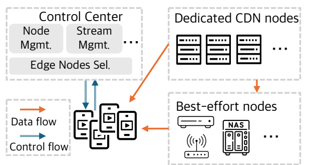
limits performance. Fortunately, *multiple best-effort nodes* that are chosen properly can complement each other for robust live content delivery. Specifically, these nodes can act as intermediaries that pull different parts of the same live stream from dedicated CDN nodes and then transmit content to clients. In this way, the performance degradation on one node can be mitigated by the other available nodes.

However, this design raises an additional challenge: determining which best-effort nodes should deliver each video session. This process is referred to as user mapping. User mapping relies on the status information (*e.g.*, bandwidth availability and forwarding streams) of each node. Given the hyperscale (*e.g.*, around 1 million) of best-effort nodes, it is impractical to use a centralized control plane to collect real-time node states (as done in LiveNet [39]). On the other hand, although such states are available at each node, a node cannot directly access the information of other nodes due to NAT constraints. Therefore, user mapping is also difficult to update in a distributed way (as done in VDN [48]). To this end, we are seeking a *collaborative control plane* that combines both centralized and distributed views.

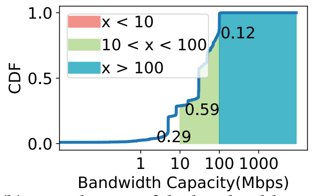
The insights above motivate the design of RLive, a robust and scalable delivery system for live streaming services. RLive aims to achieve two goals: (*i*) *Robustness*: ensuring a QoE comparable to traditional CDN-only deployments, and (*ii*) *Scalability*: expanding the system's capacity to handle increasing live streaming demand.

Nevertheless, three practical challenges arise in meeting the stringent QoE requirements of real-time generated live content. First, the global controller and the individual edge nodes update their information at different timescales, potentially leading to inconsistent user mapping decisions, which are hard to integrate. Second, clients must be able to reorder video data from multiple edge nodes into a single, continuous stream in real-time, while most live streaming protocols (*e.g.*, HLS [51] and FLV [27]) lack an explicit identifier to track the data sequence. Third, there are multiple choices for where lost data should be retransmitted (*e.g.*, at the CDN or best-effort nodes), reflecting the trade-off between recovery time and bandwidth cost. Determining the proper loss recovery choice under dynamic network conditions is non-trivial.

RLive addresses these challenges by introducing a **multi-layer collaborative control plane** as well as a **redundancy-free multi-source data plane**. Specifically, RLive facilitates a three-layer collaboration (i.e., the global controller, individual edge nodes, and user clients) to balance centralized coordination for global optimization with decentralized decision-making for local responsiveness. On the data plane, RLive splits each video stream into parallel substreams, coupled with a distributed sequencing algorithm that maintains cross-source consistency, enabling real-time reordering. In addition, RLive incorporates a QoE-driven data loss recovery mechanism that can strike the perfect balance between recovery time and cost.



(a) The workflow of client streaming from CDN with best-effort nodes.



(b) Distribution of the bandwidth capacity among best-effort nodes.

Figure 1. Live CDN with best-effort nodes.

Deployed to support several large-scale live streaming services over the past 3 years, RLive has reduced rebuffering events by 14.9–20.1% and improved video bitrate by 10.2–11.4%, compared to the traditional CDN-only delivery. Moreover, it expands the system's capacity to meet bandwidth demands by approximately 3×. These results demonstrate the effectiveness of RLive in successfully leveraging best-effort edge resources to scale live streaming services with robust data transmission and guaranteed QoE.

In summary, this paper makes three key contributions:

- Presenting RLive, a first of its kind to exploit the massive best-effort edge resources to build a robust delivery system for scaling live streaming services.
- Designing a collaborative control plane and a multi-source data transmission architecture to address the key challenges of deploying RLive at scale.
- Deploying RLive in ByteDance CDN to support several popular live streaming services. The large-scale deployment revealed improved QoE and scaling effectiveness. Our work provides practical design choices for leveraging underutilized edge resources for live streaming.

Ethics. This work raises no ethical concerns. All data and resources were used with permission and contained no private user information. It is also worth noting that the best-effort nodes function solely as relays of validated CDN content: they do not modify data and never handle user uploads or private traffic.

2 Background and Motivation

2.1 Live CDNs with Best-effort Nodes

CDN operators play a central role in delivering seamless video experiences for live streaming services by maintaining a globally distributed infrastructure. CDNs typically rely on dedicated edge nodes with high and stable bandwidth capacity. However, scaling such infrastructure to meet the ever-growing traffic demand of live streaming presents substantial challenges [78]. Although expanding capacity with dedicated resources (*e.g.*, high-performance servers) is feasible, it incurs significant cost, both in terms of infrastructure deployment [24, 75] and bandwidth usage [2, 66], which represents a considerable financial burden [83].

In response, CDN operators have begun exploring cost-effective alternatives by renting resources provided by third-party vendors worldwide [14, 26, 54]. These vendors aggregate underutilized bandwidth from geographically distributed locations, offering resources that are closer to end users than dedicated CDN nodes and often deliver competitive round-trip times (RTTs) [83]. When aggregated and integrated into CDN infrastructure, we refer to them as **best-effort nodes**.

In ByteDance, best-effort nodes act as extended relays of the CDN. Most of these nodes are deployed in ISP-managed facilities, such as base stations or apartment gateway devices. Others are installed in residential settings where users cannot control the devices or upload content via them. Deployment always involves informed consent, and participants may receive incentives (*e.g.*, app credits) for maintaining a minimum uptime. Note that, except for streaming content, no user traffic passes through these nodes.

As a major CDN operator serving several large-scale live streaming services, we are among the first to incorporate best-effort nodes into a global CDN architecture. Figure 1(a) illustrates the typical architecture of a CDN augmented with best-effort nodes, which reflects our initial system (§ 2.2). When a client requests a live video stream, it consults the control center (on the control plane) to obtain the address of a source node–either a dedicated CDN node or a best-effort node–and then initiates data transmission (on the data plane) from that node according to the retrieved address.

In our system, best-effort nodes cost 20–40% less per unit of bandwidth than dedicated nodes. However, these nodes are usually equipped with low-end computation and storage resources, resulting in *unstable availability*. In addition, many are located behind NATs of varying types, further complicating connection stability. Bandwidth capacity also varies widely: as shown in Figure 1(b), approximately 29% of these nodes have link capacities below 10 Mbps, and only 12% exceed 100 Mbps. This *limited bandwidth capacity* restricts the number of concurrent live-streaming sessions a node can support and makes it more susceptible to performance degradation under network jitter.

2.2 Failure of the Strawman Attempt

To scale our original CDN system with best-effort nodes without sacrificing performance, an intuitive way is to simply extend the CDN with an additional layer that only consists of the top 1% high-availability best-effort nodes in terms of bandwidth capacity and connectivity. We refer to this solution as *single-source transmission*. In this way, clients pull each stream from a single best-effort node, which in turn subscribes to a dedicated node for the requested content.

However, this strawman solution fails to meet the requirements for serving our 750 million daily active users due to the following inherent limitations:

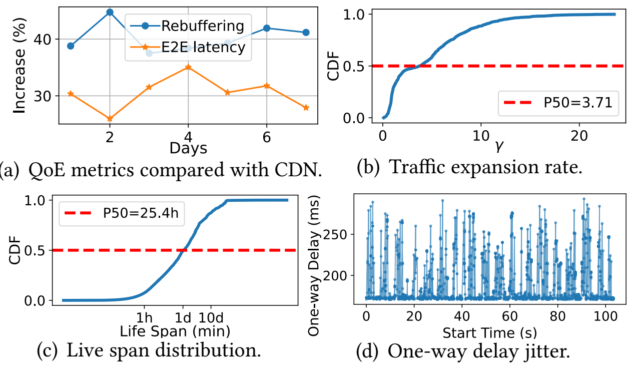


Figure 2. Characterizing best-effort nodes with single-source transmission.

(i) Degraded QoE performance. We evaluate E2E latency and rebuffering events per hundred seconds, as illustrated in Figure 2(a). The results indicate a 26%–35% increase in E2E latency and a 37.5%–44.7% rise in rebuffering events, highlighting a notable QoE degradation. These degradations arise from two factors. First, best-effort nodes are prone to experiencing significant performance degradation or temporary link failures due to their instability. Figure 2(c) shows the distribution of the live span of individual best-effort nodes. Notably, nearly 50% of the nodes have a lifespan of no more than one day, meaning they go offline at least once per day. This level of churn introduces randomness and instability, which can disrupt ongoing streaming sessions and degrade user experience. Second, their limited bandwidth capacity reduces traffic burst resilience, causing congestion-induced latency spikes that manifest as significant one-way delay jitter. Figure 2(d) illustrates an example of this phenomenon during a viewing session through one best-effort node.

(ii) Limited scaling effect. The bandwidth constraints of best-effort nodes fundamentally limit their ability to deliver live streaming services for large-scale concurrent clients. To quantify this limitation, we define the *traffic expansion rate* as $\gamma = \frac{\text{Serving Traffic}}{\text{Backward Traffic}}$, where serving traffic denotes data delivered to clients by best-effort nodes, and backward traffic represents data retrieved from the dedicated CDN nodes by the best-effort nodes. A best-effort node with γ can theoretically support $\gamma - 1$ additional clients compared to pure dedicated nodes. Figure 2(b) presents the distribution of γ during single-source transmission, revealing a median $\gamma = 3.7$, with 58.5% of nodes having $\gamma < 5$. Given that only 1% of best-effort nodes (i.e., several thousand nodes) are actively utilized, this approach yields fewer than 0.1 million (calculated based on $\gamma - 1$ of all nodes) additional clients, which is insignificant against multi-million peak concurrency.

Summary. For the successful scaling of live streaming services, it is imperative to utilize every available best-effort node. Nevertheless, our deployment experience reveals that even using the top 1% of these best-effort nodes that are expected to be the most capable, the single-source transmission system cannot ensure large-scale robust data delivery. The pressing question then arises: *How can we fully exploit all the best-effort resources, including the weak ones, while offering a competitive QoE comparable to that provided by pure CDN?*

2.3 Opportunity: Multi-Source Transmission

The root cause of the observed robustness and scaling issues in single-source transmission lies in the inherent instability and limited bandwidth of best-effort nodes. Specifically, a single node is constrained by its hardware performance. Hence, a practical solution is to leverage resources from multiple nodes simultaneously by **multi-source transmission**.

Prior work [9, 15, 28, 78] leverages multi-source delivery but relies on redundant content replication (*e.g.*, transmitting identical content across multiple links) for reliable transmission, which sacrifices bandwidth efficiency and thereby limits scalability. To solve this issue, we begin exploring a **redundancy-free robust data plane** design. Specifically, a single video stream can be divided into multiple *substreams* (*e.g.*, based on video frame boundaries) and pushed by distinct best-effort nodes. This design provides two benefits: *(i)* Transmitting data over multiple physical links makes the system robust against single-link failures by isolating bottleneck links [36, 78]. *(ii)* Since link performance degradation typically exhibits temporal and spatial locality, i.e., spanning multiple consecutive video frames [56], distributing these frames across different links is promising to mitigate the impact of such locality.

While multi-source transmission enables the effective utilization of millions of best-effort nodes, the hyperscale imposes substantial pressure on the control plane. A critical challenge lies in the user-substream mapping, which must dynamically allocate use requests to optimal edge nodes while navigating real-time network fluctuations and node instability under massive concurrency. The common practice adopts global assignment [29, 30, 42], where each user is mapped according to the centralized control plane optimizing performance and cost. This method relies on accurate and timely information about network conditions and node utilization. However, in scenarios involving unstable and hyperscale nodes, it is challenging for the global controller to promptly and reliably obtain such information, making this method impractical.

We find the opportunity lies in that different components have different levels of confidence in their network observations. In detail, the global scheduler has a broad and confident view of the static and historical attributes of all the best-effort edge nodes (*e.g.*, ISP, location, connection success rate) while clients are in the best position to observe and adapt to local network changes promptly. Mediating between them, edge nodes infer resource bottlenecks by aggregating subscriber reports. To this end, we aim to construct a **collaborative control plane**, allowing multi-layer components to make mapping decisions based on their unique views.

Table 1. Live streaming service overview.

	6 am	12 pm	6 pm	12 am	max
#streams	~0.70M	~1.60M	~1.75M	~1.38M	~2.47M
#nodes	~0.9M	~0.95M	~1.0M	~1.02M	~1.05M

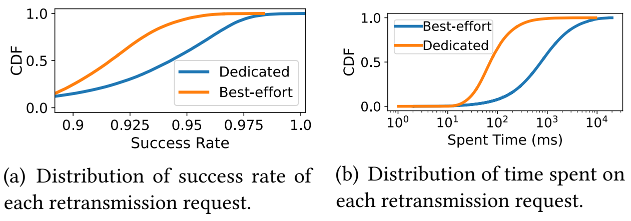


Figure 3. Retransmissions comparison.

2.4 Design Challenges

Although our two key insights provide clear guidance for the system design, we encounter three key practical challenges when implementing them.

Temporal misalignment in user mapping. During the user-substream mapping ("user mapping" for short) process, the distinct layers within the control plane update their perspectives at different timescales. The global controller passively updates node status at the *second* level, relying on reports from edge nodes. In contrast, client controllers monitor real-time network conditions at the *millisecond* (packet) level. Unlike the above, edge advisers synchronize the status of subscribers at a *hundred-millisecond* scale. The variance in information update timescales results in discrepancies in end-user mapping decisions across different layers, posing a significant challenge in integrating these decisions.

Resilient substream ordering at massive scale. Clients need to reassemble the received video frames from multiple substreams into a single stream in real time, relying on information about the frame sequence number. However, mainstream live-streaming protocols such as HLS and FLV lack such identifiers. Although it is possible to generate and record frame sequences on an additional central server, this method has scalability and fault tolerance issues. The vast computational load from a large number of concurrent streams (2.47M as presented in Table 1) and the potential loss of sequence information during server failures introduce delays and rebuffering. Consequently, ensuring efficient and robust frame reordering under high concurrency is a significant challenge for supporting multi-source transmission.

Cost-reliability trade-offs in retransmission. Packet loss impacts the QoE performance of video streaming [6, 46, 56]. One general solution is to retransmit the lost packets from the sender, i.e., dedicated CDN nodes or best-effort nodes. However, optimizing loss recovery efficiency while minimizing retransmission cost is complex due to the heterogeneous characteristics of best-effort nodes and dedicated nodes. Figure 3 depicts the distribution of retransmission success rates and the time spent on retransmission requests sent to dedicated and best-effort nodes. The results show that

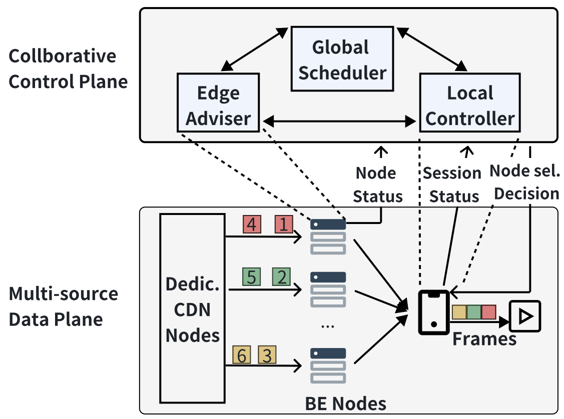


Figure 4. The High-level Design of RLive.

dedicated nodes achieve higher retransmission success rates and shorter retransmission times due to their reliability, but incur higher bandwidth expenses. Specifically, the median retransmission time is 71.1ms for dedicated nodes compared to 778ms for best-effort nodes, and the success rate is 94.09% versus 91.44%, respectively.

3 RLive Overview

To address these challenges, we introduce RLive, consisting of a multi-layer collaborative control plane and a multi-source multi-substream transmission in the data plane. Figure 4 presents the overview of RLive.

The control plane ensures **responsiveness and dynamism** by facilitating collaboration among three layers of components: the global scheduler, client controller, and edge adviser. The whole mapping involves two main processes: hybrid user mapping control and real-time node switching. During each process, the global scheduler is responsible for recommending nodes based on its global view, and real-time decision-making is delegated to client controllers, which dynamically adjust node selection based on direct observations of network conditions. Meanwhile, the edge nodes provide proactive suggestions when they detect QoS degradation or resource under-utilization.

In the data plane, RLive segments each live stream into multiple substreams at the frame level. Best-effort edge nodes are equipped with a **distributed** frame sequencing algorithm and deliver the local sequences alongside data packets of substreams to clients. The clients then combine the local sequences into global sequences to ensure the data retrieved from various nodes is reorganized in the correct order for seamless playback, eliminating any single point of failure and guaranteeing scalability. Additionally, we incorporate an **QoE-driven** loss recovery mechanism, which introduces a probabilistic model that captures the success probability of different loss recovery strategies before the playback deadline of frames, enables intelligent selection of the optimal

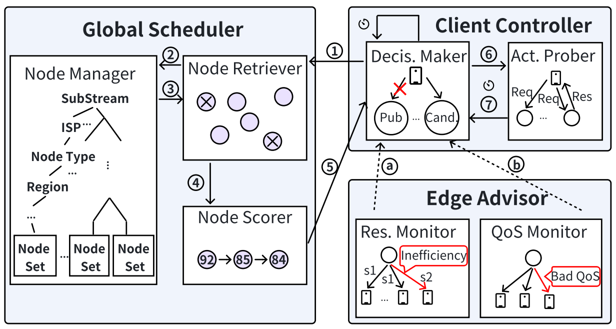


Figure 5. Overview of the collaborative control plane.

source for recovering lost data based on real-time assessments of buffer size and network conditions.

Together, RLive enables adaptive control under largescale dynamics and ensures timely, in-order delivery on top of heterogeneous and unreliable edge conditions, thereby meeting the stringent QoE requirements of live streaming.

4 Collaborative Control Plane

This section presents RLive's control plane, which comprises three levels of controllers with distinct views that collaboratively perform user mapping. As shown in Figure 5, it handles two key tasks: initial mapping at client playout startup (§4.1) and real-time switching in response to QoS degradation or under-utilized best-effort nodes (§4.2).

4.1 Hybrid User Mapping Control

When a client initiates a stream, two concurrent tasks begin: pulling the full stream from the original CDN to fill the initial playout buffer of the client's player and identifying best-effort nodes for multi-source data transmission. This parallelism ensures low first-frame latency. The process of best-effort nodes identification starts with candidate recommendations from the global scheduler, followed by client-driven fine-tuning via active probing.

4.1.1 Global Candidates Recommendation. The global scheduler faces the challenge of managing a large-scale network with limited real-time visibility. To avoid noise from outdated or highly dynamic features, it focuses on two categories: (*i*) Static Features, including inherent attributes such as location, ISP, node type (i.e., whether a high-quality node or not), and connection type. (*ii*) Temporal Features, including attributes like bandwidth utilization and connection success rate, which vary across video frames in each session. To track these features accurately, nodes send lightweight updates (about 150 bytes) to the scheduler every 5s when active (i.e., forwarding streams) and every 10s when idle, striking a balance between freshness and scalability across millions of nodes.

The goal of the global scheduler is to identify a candidate list of K nodes that maximize availability while minimizing cost:

$$\arg\max_{|C|=K} \sum_{i\in C} \frac{a_i}{p_i},$$

where a_i and p_i represent the availability and cost of node i. If a node is already forwarding the sub-stream, the cost is limited to the bandwidth consumed by the client; otherwise, additional back-to-CDN traffic (i.e., the substream node requests the CDN for the transmission of the requested data) incurs extra cost.

However, this optimization faces two challenges. Firstly, top-K recommendations across a large node set are computationally expensive. Secondly, node availability depends on the requested sub-streams and the clients as well, requiring individualized estimation for the clients. To address these challenges, the global scheduler first retrieves candidate nodes using a tree-based hash structure that filters them by static features. It then applies a personalized scoring mechanism tailored to each client, ranking the candidates for every request based on client-specific and temporal factors such as proximity, access type, and historical connection success rates, thereby prioritizing nodes most likely to deliver better performance for that client.

The tree-based hash structure aims to support efficient and priority-aware node retrieval. At each layer of the tree, specialized hash functions are used to map attribute values, ensuring fast and precise navigation. The retrieval process begins by seeking exact matches, traversing from root to leaf nodes along the full attribute path (stream → ISP → node type → ...). If the number of retrieved nodes is insufficient, the matching criteria are progressively relaxed in reverse order of priority, allowing for a broader search that still yields relevant nodes when an exact match cannot be found.

Each candidate node is then assigned a score reflecting its availability for the specific request:

$$S(n_i, c) = \alpha_1 * N(n_i, c) + \alpha_2 * G(n_i, c) + \alpha_3 * R(n_i, c) + \alpha_4 * B_i,$$

where n_i and c represent the candidate node and client respectively. We summarize the key factors that influence the availability as follows:

- Same-network Preference ($N(n_i, c)$): Nodes in the client's local network (i.e., the same BGP prefix) receive priority.
- Proximity ($G(n_i, c)$): Geographically closer nodes are prioritized to minimize latency.
- Bandwidth Availability (B_i): Nodes with higher residual bandwidth are favored to ensure stable throughput.
- NAT-Specific Success Rate ($R(n_i, c)$): Nodes with NAT type with historically higher connection success rates are weighted more heavily.

Note that the weights (α) are different across platforms (*e.g.*, Android and iOS) and applications. The candidate nodes are then sorted by their scores, and the top-K nodes are forwarded to the client for final selection.

4.1.2 Local Fine-Tuning.

Once the client receives the candidate node list from the global scheduler, it begins the fine-tuning phase. During this phase, the client actively probes each candidate node to collect real-time performance data, focusing on dynamic and unstable features. This probing captures the network's immediate performance from the client's perspective, allowing it to evaluate the current state of each candidate.

Specifically, the client sends a connection request for the sub-stream to each candidate node. The first node to respond successfully is chosen. Note we send application-level connection attempts, rather than Ping, to gauge not only the latency, but also the available bandwidth capacity. In practice, we limit probing to at most three candidates, as A/B testing shows that probing additional nodes yields marginal improvements (<1%) in success rate while incurring extra bandwidth and computation overhead linearly.

4.2 Real-Time Edge Node Switching

The inherent instability of best-effort nodes and dynamic session conditions necessitate frequent end-user re-mapping to ensure both robust data transmission and maximal node utilization. RLive employs a dual-control design for real-time edge node switching. The client controller periodically probes the QoS metrics of both the current publisher nodes and candidate nodes to identify opportunities for improved performance, serving as a client-side control. Simultaneously, the edge monitors for significant QoS degradation or resource underutilization, which will trigger immediate edge node switching, providing proactive suggestions. This dual-control design balances optimization with proactive adaptation, ensuring minimal service disruption while maintaining high resource efficiency in dynamic environments.

4.2.1 Client-side Control.

The *client controller* keeps a focus on maintaining high QoS by continuously monitoring network conditions, especially in terms of RTT. This mechanism operates on a scheduled basis, enabling clients to periodically assess their current publishers against the list of candidate nodes recommended by the global scheduler. The objective is twofold: to ensure rapid failover upon sudden QoS degradation or edge node outages, and to maintain steady optimization under stable network conditions. The switching logic relies on the following condition to determine if a switch should be initiated:

$$RTT_{\text{cur}} > \min_{i\in\text{Candidates}} (RTT_i + t_{\text{change}}),$$

where RTT_{cur} and RTT_i means the RTT between the client and the publisher node and the candidate node i respectively; t_{change} accounts for the switching costs, including reconnection and initialization delays. Note that the candidate nodes are recommended and updated by the global scheduler.

When this condition is met, the client identifies a better candidate node (i.e., the node with the lowest RTT among the candidates) and initiates a switch. This process immediately enhances QoS by transitioning to a better-performing node.

4.2.2 Proactive Suggestions.

The best-effort nodes play a pivotal role in enhancing both the cost-efficiency and the QoS by acting as an active adviser that complements the client-side control. Specifically, this proactive control is driven by two distinct objectives: cost-efficiency and QoS.

Cost-aware Trigger. Best-effort nodes continuously monitor their resource utilization to identify potential inefficiencies. When the sliding average of resource utilization, $\bar{\eta}_{\text{node}}$, drops below a predefined threshold (θ), the system detects underutilization and recommends a switch to reduce costs. Additionally, the best-effort node contacts the global scheduler to double-check whether the average utilization of the forwarding stream ($\bar{\eta}_{\text{stream}}$) also falls below the threshold. If both conditions are met, the best-effort node proactively suggests a switch to the client. This mechanism reduces back-to-CDN traffic by redistributing requests away from underutilized nodes, thereby saving costs. In our deployment, nodes re-evaluate their utilization every 10 seconds and only advertise themselves to clients when underloaded. This design maintains utilization above 60% for over 70% of active nodes and scales to millions of nodes in deployment.

QoS-Aware Trigger. It tackles scenarios where link issues impact individual clients. A best-effort node may have high QoS for most, but a specific client can face degraded performance. The node, having a broader QoS view, can spot potential degradation before client-side control. Specifically, best-effort nodes calculate the Z-score of each connection's QoS metric relative to the node's overall connections to detect such cases. Formally, the best-effort nodes operate as follows: (*i*) The QoS metric for each client connection is evaluated, and its Z-score is computed: $z_i = \frac{x_i - \mu}{\sigma}$, where x_i is the QoS metric of connection i, and μ and σ represent the mean and standard deviation of QoS metrics across all connections, respectively. (*ii*) Connections with Z-scores in the top 5%, indicating significant deviations from the average, are flagged as potential outliers. (*iii*) When these outliers are detected, the node proactively sends edge node switching suggestions to the affected clients. This approach ensures that isolated link-quality issues are addressed on time.

Notably, once a client receives a switch suggestion from the publisher, it immediately enters the client-side control to evaluate if better nodes are available for switching. If no better node is identified, the switch suggestion is not executed to avoid potential QoS degradation. In this case, the client requests the global scheduler to update the candidate node list for potentially better candidates.

5 Robust Data Transmission

RLive divides each live stream into multiple sub-streams , which are delivered by best-effort edge nodes. It ensures robust data transmission under scale through two key mechanisms: (*i*) distributed frame sequencing that leverages the computational capabilities of best-effort nodes to reorder data accurately, and (*ii*) a QoE-driven loss recovery strategy that dynamically selects the optimal timing and source for retransmission.

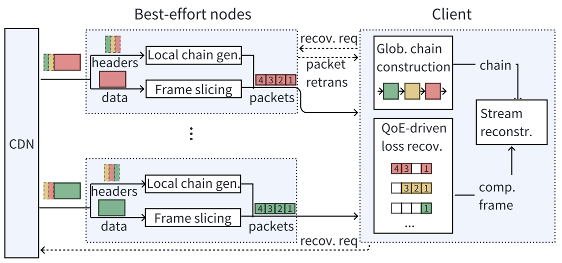


Figure 6. Distributed transmission overview.

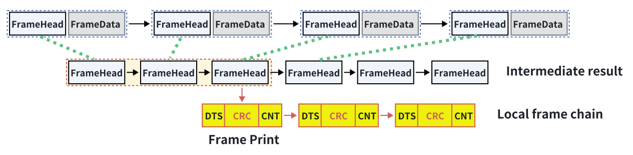
5.1 Multi-source Multi-substream Transmission

After end-user mapping determines substream sources, clients send subscription requests to the assigned best-effort nodes. Upon receiving a request, an best-effort node adds the client to its substream-specific subscriber list. If the node is not already forwarding the requested substream, it subscribes to the CDN, which delivers the stream as compressed Network Abstraction Layer Units (NALUs) encoded using standard codecs (i.e., H.264/AVC or H.265/HEVC). Each NALU encapsulates a complete video frame or a decodable slice. For simplicity, we refer to NALUs as frames throughout the paper.

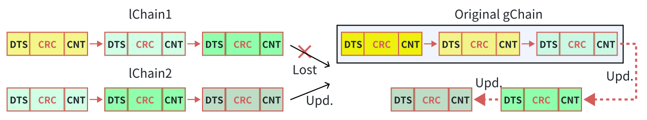
Figure 6 illustrates the overview of the multi-source multi-substream transmission process. Traditional chunk-based delivery (*e.g.*, multi-second segments in HLS) incurs higher end-to-end latency and potential head-of-line blocking, as best-effort nodes must first accumulate the entire chunk before forwarding it to the client [41, 52]. In response, RLive adopts fine-grained frame-level transmission to improve responsiveness and reduce latency. However, fine-grained delivery also introduces new challenges: frames are scattered across nodes and must be accurately reordered at the client to ensure smooth playout.

To address this, RLive introduces a frame chain abstraction to ensure in-order reconstruction of frames across heterogeneous nodes in a distributed way. Specifically, the CDN provides *complete* frames for the requested substream and *headers* of other substreams within the same stream. Upon receiving a complete frame, the best-effort records the associated frame header and updates its local frame chain. Meanwhile, the node segments the frame into fixed-size packets, embeds the local frame chain, and pushes packets sequentially to subscribers via UDP. This ensures in-order delivery in the absence of packet loss.

At the client side, once all packets for a frame are received, the client checks whether the frame can be added to the ordered frame buffer, which is continuously consumed by the



(a) best-effort nodes-driven local frame chain generation.



(b) Clients-driven local chain combining.

Figure 7. Illustration of the distributed scalable frame sequencing algorithm.

player, based on the global frame chain it maintains, which is updated by aligning distributed local chains (§5.2). Accordingly, complete frames are buffered in sequence for playout, while missing data triggers QoE-aware recovery(§5.3).

5.2 Distributed Frame Sequencing

We present a novel, scalable frame sequencing algorithm that distributes the computation and recording of frame sequences across distributed edge devices. This design eliminates potential bottlenecks and scales seamlessly with the increasing number of streams, enhancing the robustness of frame sequencing. Specifically, RLive allows best-effort nodes to distributively generate local frame sequences, which are transmitted alongside regular data packets. Clients then use these local sequences to construct and continuously update the global frame sequence across all substreams. We illustrate the process of local frame sequence generation and global chain updating in Figure 7.

Local frame sequence generation. We design a lightweight frame footprint to uniquely identify frames and chain them for global sequence notification. Using the hash of full frame data for uniqueness requires best-effort nodes to pull all frames from different sub-streams, increasing cost and E2E latency as the footprint can only be calculated after the whole frame is received. Thus, we introduce a novel frame footprint using only frame headers, designed to ensure both the uniqueness of each frame and the ability to validate the frame sequence.

As shown in Figure 7(a), the footprint comprises three components: decoding timestamp (dts), a CRC value embedding current and prior two headers to avoid conflicts, and CNT for packet count. This method focuses on frame headers rather than entire frame data, striking a balance between efficiency and uniqueness.

When forwarding packets to subscribers, the best-effort nodes are responsible for adding local frame chains, formally $\text{footprint}_{i-\delta} \Rightarrow \text{footprint}_{i-\delta+1} \Rightarrow \ldots \Rightarrow \text{footprint}_{i-1}$, into the packets of the frame i. Specifically, we set the length of the frame chain (δ) as 4.

Combining local frame chains. Upon receiving a local frame chain from an best-effort node, the client attempts to integrate the local chain into the global frame chain. We illustrate the combining process in Figure 7(b), where the client maintains a global chain with a length of 3, and two best-effort nodes transmit two different local chains. Though one of the local chains is lost, the client can successfully combine the second chain into the global chain.

We summarize the algorithm that tries to match a single local chain on the global chain in Algorithm 1. The client first verifies continuity by checking if *lChain* contains the terminal frame of *gChain* (lines 2-8). Upon successful alignment, unmatched frames from *lChain* are appended to *gChain* with `UNLINKED` status. Each appended frame undergoes a CRC to validate that the order of *gChain* is consistent with what is recorded in the frame footprint. Frames that pass validation are marked as `LINKED` (lines 15-17). Any validation failure triggers the removal of all `UNLINKED` frames from *gChain*, preserving chain integrity (lines 19-21).

The client also manages a list of under-matched chains (*misMatchChains*), which have not been matched with the global chain due to the absence of preceding frames. When *gChain* expands via a successful merge, the client re-evaluates *misMatchChains* entries against the updated *gChain*, removing fully integrated or now-compatible chains.

Algorithm 1: Try on Matching Chain

```
1  $maxLinkedFrame \leftarrow$ LastElement($gChain$);
2  foreach $frame \in lChain$ do
3      if $findCont$ then
4          $gChain$.push_back(($frame$,UNLINKED));
5      end
6      if $frame = maxLinkedFrame$ then
7          $findCont \leftarrow$ True;
8          $expectLinkFramefootprint \leftarrow$ NextFrame.footprint ;
9      end
10 end
11 if not $findCont$ then
12     return False;
13 end
14 while not the end of $gChain$ do
15     if $expectLinkFramefootprint \in dataPool$ and the CRC
         validation succeed then
16         set the link status as LINKED;
17         $expectLinkFramefootprint \leftarrow$ next frame $\in gChain$;
18     end
19     else
20         Push out the unlinked frames from $gChain$;
21         return False;
22     end
23 end
24 return True;
```

5.3 QoE-driven Sub-stream Recovery

When data loss occurs, the system must decide whether to retrieve missing packets from either the cost-efficient best-effort nodes or the more reliable dedicated nodes. Our idea is to prioritize best-effort nodes for recovery when the lost data can be successfully retrieved before the frame buffer is exhausted, thereby minimizing bandwidth costs. However, if the buffer nears depletion and best-effort recovery risks failure, the system seamlessly switches to the dedicated ones to prevent interruptions. We modeled data loss recovery as a state-aware decision-making framework, with a probabilistic loss function that integrates bandwidth cost, the probability of frames being unplayable, and the impact of frame loss.

Action Space. For a retransmission list (i.e., consisting of the incomplete frames) with m frames, actions form an m-dimensional vector $\vec{A} = (a_1, a_2, ..., a_m)$ with four recovery options per frame:: (*i*) *Packet Recovery from best-effort nodes ($a_i = 0$).* We employ both timeout-based and fast retransmission mechanisms. Out-of-order packets indicate data loss and the client promptly triggers a fast retransmission request. Otherwise, the client initiates a standard timeout-retransmission process. (*ii*) *Frame recovery from dedicated nodes ($a_i = 1$).* Clients can also promptly retrieve missing frames from a dedicated node while subsequent frames continue to be pulled from best-effort nodes. (*iii*) *Sub-Streams switching back to dedicated nodes ($a_i = 2$).* If the transmission path of a specific sub-stream suffers from poor QoS, resulting in the loss of multiple consecutive frames, repeatedly requesting individual frames from the dedicated node becomes inefficient. To address this, we enable the client to switch the publisher of the affected sub-streams, allowing it to pull the sub-streams directly from the dedicated nodes. (*iv*) *Pulling the Full Stream from dedicated nodes($a_i = 3$).* In extreme cases, we allow the client to pull the entire stream from dedicated nodes, ensuring reliable delivery when other recovery methods fail to maintain acceptable QoS.

State Representation. To decide which action to take, we mainly focus on two primary categories of metrics: buffer states (i.e., whether the buffer is too small to maintain smooth playout) and network states (i.e., whether the QoS is too poor). Specially, we define the state

$$S = (\vec{\tau}, \vec{s}, X_{succ}, X_{fail}, L).$$

τ (m-dim) indicates the frame playout deadlines. s (m-dim) where s_i represents the data size of frame i. $X_{succ} = \{(x_{succ,i}, n_{succ,i})\}$ records the number of successfully retransmitted packets within frame i ($x_{succ,i}$) and the total number of retransmissions ($n_{succ,i}$), whereas $X_{fail} = \{(x_{fail,i}, n_{fail,i})\}$ records the number of packets that have not been successfully retransmitted yet within frame i ($x_{fail,i}$) and the total number of retransmissions ($n_{fail,i}$). L records the time taken to retrieve frames from dedicated nodes.

Loss Function. To evaluate the efficiency of each action, we adopt a probabilistic loss function that integrates bandwidth cost, the probability of frames being unplayable, and the impact of frame loss:

$$Loss(A) = cost(A) + \lambda \sum_{i=1}^{m} P(F_i|a_i, S) * risk(F_i).$$

The first item, $cost(A)$, measures the bandwidth cost introduced by action A and the second item measures the impact of executing action A under state S on the smoothness of video playout. For each frame, we evaluate the probability that not all of its packets have been received before its playback deadline ($P(F_i|a_i, S)$), as well as the risk of the frame being unplayable ($risk(F_i)$). Due to the reference dependency between video frames, the loss or delayed arrival of packets may cause the subsequent video frames to fail to be decoded successfully. Accordingly, the risk of losing an I-frame is significantly more severe than that of other frames, as I-frames can decode multiple other frames. In our implementation, $risk(F_i)$ is only related to the frame type, and a higher constant value is set for the I-frames.

Given the state S and action a_i, we model the probability respectively for best-effort nodes ($a_i = 0$) and dedicated nodes ($a_i \geq 1$). For dedicated nodes, which retransmits entire frames in single attempts with stable transmission time, the failure probability is derived from historical latency records L as:

$$P(F_i \mid a_i, S) = 1 - F_N(\tau_i), a_i \geq 1$$

, where $F_N(l) = \frac{1}{N} \sum_{l_i \in L} \mathbb{I}(l_i \leq l)$ represents an empirical distribution function (EDF) for the time spent on retransmission from dedicated nodes. For best-effort nodes, which requires packet-level retransmissions, failure probability depends on the per-packet success rate $p = \frac{x_{succ,i}}{n_{succ,i}}$, the number of missing packets $x_{fail,i}$, and feasible retries within τ_i:

$$P(F_i \mid a_i = 0, S) = 1 - (1 - (1 - p)^{\frac{\tau_i}{T}})^{x_{fail,i}}$$

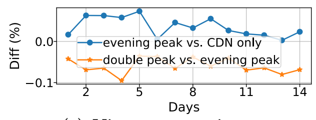
Decision. After the loss function is determined, we decide which action to execute in a straightforward way. In other words, we select the action with minimum loss:

$$\text{Final Action} = \arg\min_{A} Loss(A)$$

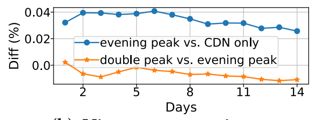
6 Implementation

Best-effort Nodes Control. The best-effort nodes used in our system are provided by third-party vendors as bare metal machines running Linux, over which we have full root access–including control over the kernel, network stack, and runtime environment. This level of access allows us to deploy lightweight containerized agents and monitoring tools, and to support multiple co-located services via virtualization. Such flexibility enables fine-grained scheduling, real-time resource management, and seamless integration with the global control plane.

Frame Delivery. A live stream comprises a continuous sequence of frames, formally represented as $f_0, f_1, f_2, ...$ For generality and simplicity, we adopt a static round-robin partitioning method according to the dts field to segment the stream into K substreams as follows: $ssid(f_i) = \text{Hash}(\text{dts}(f_i)) \bmod K$. To prevent several consecutive large frames from being assigned to the same substream and causing bursty traffic, we employ the FNV-1a hash function to ensure a uniform distribution.



(a) Views comparison.



(b) Viewers comparison.

Figure 8. Views and viewers participating in A/B tests.

Minimal Modifications on the CDN Side. In our system, the changes to CDN include: (*i*) supporting forwarding both substreams and full streams. (*ii*) supporting frame-level recovery indexed by dts. Accordingly, the CDN-side implementation demands fewer than 100 lines of code modifications in our system, primarily adding lightweight `dts`-based routing logic, ensuring compatibility with both legacy and modern CDN frameworks, thus guaranteeing broad generalization.

Subscribe-Push Streaming for Low Latency. We adopt a Subscribe-Push paradigm to minimize latency, aligning with prior low-latency designs [39, 78]: clients subscribe substreams to best-effort nodes, which then segment content into fixed-size packets and push them immediately without performing traditional control functions (*e.g.*, loss detection or congestion control).

7 Experiments

RLive has been deployed in ByteDance CDN with approximately one million best-effort nodes since 2021, enabling the scaling of multiple mainstream live streaming applications, such as Douyin. In this section, we evaluate RLive's performance in our production environment. We first present the results of A/B tests when RLive is approved to be online, comparing the performance with dedicated CDNs. Then we evaluate RLive with a single-source transmission scheme to prove its enhanced robustness and scalability. Finally, we conduct a deep dive on the components of RLive.

7.1 Large-scale A/B Tests

7.1.1 Setting. We conducted two A/B tests during separate time periods, each encompassing 100% of Douyin Lite android users, grouped by their unique IDs.

Evening Peak vs. Dedicated CDN Only. In the first A/B test, users in the control group accessed full streams directly from dedicated CDN nodes, while users in the test group were allowed to pull streams through RLive during evening peak hours (8 p.m.–11 p.m.). Notably, users in the test group initially retrieved full streams from the CDN nodes. The system switched to RLive only when the following conditions were met: (*i*) The stream popularity exceeded a predefined threshold to maximize the benefits of multi-substream transmission. (*ii*) Users' viewing times exceeded a certain threshold (30 seconds in our deployment) to avoid frequent switches.

Double Peak vs. Evening Peak. After the successful deployment of RLive during evening peaks, we hypothesized that extending its usage to noon peaks would yield additional benefits. Thus, in the second A/B test, users in the control group accessed streams as described for the test group in the first A/B test. In contrast, users in the test group were permitted to pull streams through RLive during both noon peaks (11 a.m.–2 p.m.) and evening peaks. All other configurations were identical to the first A/B test.

Each A/B test lasted two weeks, involving **billions of views daily**. Figure 8 shows the differences between the test and control groups in terms of view count and viewer count for both tests. The observed differences are on the order of 0.01%, demonstrating the fairness of these A/B tests.

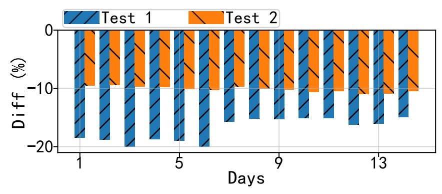
7.1.2 Robust QoE Performance. We evaluate RLive by analyzing QoE metrics recorded on client devices. Figure 9 illustrates the rebuffering times per hundred seconds, video bitrate, and end-to-end latency. Each subplot shows the daily-averaged differences between the test and control groups in the two A/B test sets for these metrics.

First, Figure 9(a) compares the rebuffering times between the two groups, with differences expressed as percentages (calculated relative to the control group). The results demonstrate that allowing clients to pull streams through RLive significantly reduces rebuffering times. During evening peaks, rebuffering times decreased by approximately 15% when clients utilized RLive. This improvement is particularly impactful as CDN servers often encounter bandwidth bottlenecks during peak hours [78], potentially exacerbating rebuffering events. By leveraging best-effort nodes, RLive successfully scales the existing CDN and alleviates CDN bandwidth pressure, while its multi-source, multi-substream mechanism ensures stable and reliable data delivery.

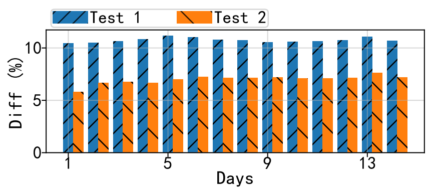
Additionally, a further 10% reduction in rebuffering times was observed when clients were allowed to pull streams through RLive during both evening and noon peaks. This moderate improvement suggests that, while noon rush hours impose bandwidth pressure, the strain on CDN resources is less severe compared to evening peaks. Consequently, clients in the control group experienced fewer rebuffering events during noon peaks, resulting in slightly reduced gains from RLive during this period.

Second, Figure 9(b) presents the comparison of video bitrate. A similar trend to rebuffering times is observed: during evening peaks, the bitrate increases by approximately 10.5% when clients use RLive, while it rises by around 7% when clients use RLive during both noon and evening peaks. This indicates that RLive helps prevent the CDN from reducing video bitrate to accommodate more viewers during rush hours, ensuring a better-quality streaming experience.

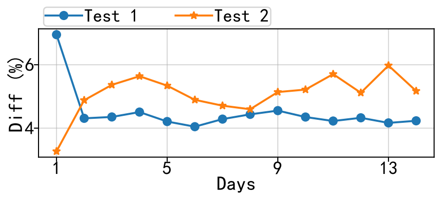
Third, Figure 9(c) highlights the end-to-end latency and we can see a 4–6% increase in latency in both tests. This is expected, as adopting RLive introduces an additional processing time between the source and the client, and time is required to reconstruct multiple substreams. Fortunately, the increase remains well below the acceptable threshold for



(a) Rebuffering times per hundred seconds.



(b) Video bitrate comparison.



(c) End-to-end latency comparison.

Figure 9. A/B test results.

Table 2. Equivalent traffic(EqT) reduction. Test 1: Evening Peak vs. CDN Only; Test 2: Double Peak vs. Evening Peak.

	Test 1 (%)	Test 2 (%)
Evening equivalent traffic	7.99± 0.36	0.13 ± 0.09
Non-peak equivalent traffic	0.02 ± 0.26	6.16 ± 1.63

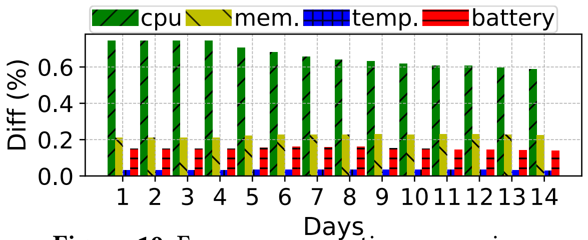


Figure 10. Energy consumption comparison.

live streaming (2s for ultra-low latency requirements [57]). Nevertheless, this observation underscores the importance of balancing rebuffering reductions with potential latency trade-offs. Effective latency management remains critical for optimizing user experience in live streaming scenarios.

7.1.3 Cost Reduction. We utilize equivalent traffic (EqT) to evaluate the cost reduction achieved by RLive. EqT is defined as the product of the normalized unit cost of resources and the volume of traffic, offering a direct measure of the traffic saved. In our A/B tests, EqT is used in place of bandwidth cost, as some edge resources are billed based on their monthly percentile bandwidth, making it difficult to directly assess the impact on final costs within the short duration of the tests.

Table 2 shows the EqT comparisons between the test and control groups across the two A/B tests, respectively. The results indicate a 7.99% reduction in evening EqT during the first test and a 6.16% reduction in non-evening EqT during the second test. These findings underscore RLive's ability to significantly reduce bandwidth costs by leveraging lower-cost best-effort nodes. A detailed comparison with the single-stream scheme, proving the scalability of RLive, will be provided in Section 7.2.

7.1.4 Energy Consumption. In addition to evaluating QoE performance and bandwidth cost, it is crucial to assess the energy consumption of RLive on the client side. Figure 10 reports metrics including CPU and memory usage, temperature, and battery consumption of the clients. The results show that RLive incurs a marginal increase of 0.58%–0.74% in CPU, 0.21%–0.22% in memory, 0.02%–0.03% in device temperature, and 0.13%–0.15% in battery consumption.

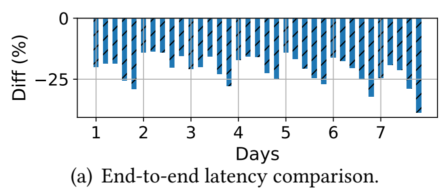
To minimize these increases, we optimized the implementation of RLive by improving data structures, reducing redundant copy operations, and enhancing the data transmission protocol to eliminate unnecessary data exchange. Consequently, the energy consumption impact is minimal, and no negative feedback related to energy usage was received from users during the A/B tests.

7.2 Multi-source Multi-substream Transmission

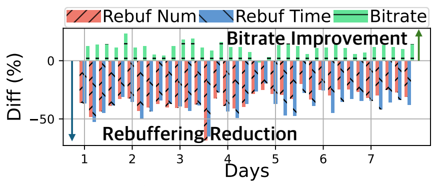
7.2.1 Settings. Our system categorizes best-effort nodes into two tiers: Best-effort nodes with limited bandwidth/stability employ RLive's multi-source multi-substream transmission (Multi), while high-capacity nodes deliver single-source streams (Single) directly from CDN edges. Despite a 10:1 traffic volume ratio (Multi vs. Single), both approaches leverage best-effort nodes with an additional network hop to the original CDN. This setup enables a direct comparison of Multi's robustness and scalability against Single.

7.2.2 Robustness Enhancement. As previously detailed in Section 2.2, within a network environment characterized by weaker stability, Multi is likely to exhibit greater resilience to jitter and potentially lower latency. We gathered statistics during peak hours (7 p.m.–11 p.m.) over the course of a week, from December 23rd to December 29th, in a production environment. Figure 11(a) presents the average E2E latency difference between Multi and Single. The results indicate that Multi attains a 12%–30% reduction in latency, thereby validating its real-time performance capabilities. Furthermore, Figure 11(b) showcases a comparison in terms of rebuffering times per hundred seconds, rebuffering duration per hundred seconds, and bitrate. The data reveals a substantial reduction in rebuffering occurrences and an improvement in bitrate, highlighting the enhanced robustness of RLive.

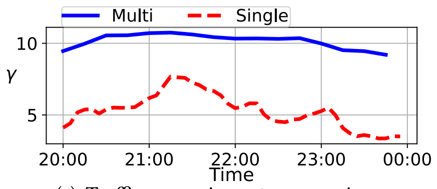
7.2.3 Scalability Analysis. In Section 2.2, we also pointed out the limited traffic expansion rate. We posit that the adoption of Multi can enhance this ratio, and consequently, improve the ability to scale CDNs. Figure 11(c) depicts the traffic expansion rate in the peak hours for one day and we see that Multi transmission has nearly doubled the traffic expansion rate. Additionally, since the bandwidth capacity of best-effort nodes has not reached saturation, and during scheduling, we tend to allocate nodes with high connection success rates and those within the same local area network, rather than



(a) End-to-end latency comparison.

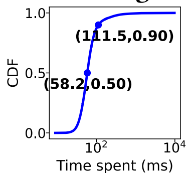


(b) QoE comparison.

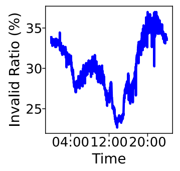


(c) Traffic expansion rate comparison.

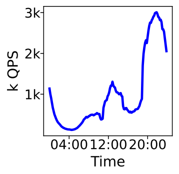
Figure 11. Evaluation on multi-source (Multi) compared with single-source transmission (Single).



(a) Time spent for node recommadation.



(b) The percentage of invalid nodes.



(c) QPS of the global scheduler.

Figure 12. Global control plane statistics.

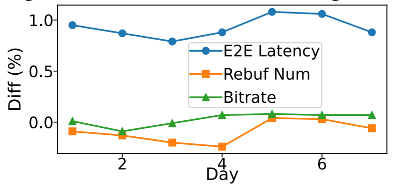


Figure 13. A/B test result under RTM.

Table 3. Centralized vs. distributed frame sequencing

	Retransmission Rate(%)	Rebuffering Times	Rebuffering Duration(ms)
Red. (%)	25.50	3.49	5.96

Table 4. FIFA case study: QoE of RLive vs. CDNs

	#Views	Bitrate	E2E lat.	Rebufferings
Diff. (%)	+21.78%	+1.72%	-4.75%	-8.82%

solely aiming to maximize the traffic expansion rate. As a result, when meeting more demands, there remains significant potential for further improving the traffic expansion rate under Multi transmission.

7.3 Component-wise Evaluation

7.3.1 Control Plane Evaluation. Due to the large number of viewers, the QPS (Queries Per Second) of the global scheduler reached several million during the evening peak period (as shown in Figure 12(c)). Under such pressure, providing responsive edge node recommendations becomes a challenge. Figure 12(a) illustrates the distribution of the time spent on node recommendations, showing that the median time is 58.2ms, with the 90th percentile value at 111.5ms. This relatively low response time reflects the efficiency of the simple strategy employed by the global scheduler. Note that clients simultaneously initialize their stream pulls from the CDN, which is fast to connect, and query candidate nodes from the global controller.

However, as shown in Figure 12(b), up to 35% of the recommended nodes may be invalid—either due to poor quality or have reached their capacity, prompting clients to make finer-grained decisions based on their local views.

7.3.2 Frame Sequencing Evaluation. During the evolving iterations, we adopted a centralized approach for frame sequencing. This approach works as follows: Firstly, we utilize several high-quality best-effort nodes, which we refer to as "super nodes". These super nodes pulled full streams from the CDN edge, segmented them at the frame level, and computed CRC values to establish sequence orders. Other best-effort nodes queried super nodes periodically to embed these orders into data packets. However, this method proved unscalable and fragile. As concurrent streams grew, super nodes became bottlenecks, requiring constant expansion. Additionally, super node failures caused significant delays in recovering sequence chains, degrading system reliability.

In contrast, our distributed frame sequencing method eliminated these issues. By enabling best-effort nodes to independently generate sequence orders based on the frames they transmit, we ensured synchronization between data packets and sequence orders without relying on super nodes. This approach not only improved robustness but also enhanced performance. As shown in Table 3, the distributed method reduced the retransmission rate by 25.5%, while rebuffering frequency and duration per hundred seconds decreased by 3.49% and 5.96%, respectively. These results clearly demonstrate the superiority of the distributed approach in scalability, reliability, and overall QoE.

7.3.3 Case Study: 2022 FIFA World Cup. We further validate the scalability of RLive through a case study of the 2022 FIFA World Cup, held from November 20 to December 18, 2022. We select this event because mega-scale live broadcasts naturally stress delivery systems with massive, short-term bandwidth surges, which are difficult to handle by provisioning additional dedicated CDN capacity due to the time and cost required. In contrast, RLive can rapidly mobilize existing best-effort resources to meet such bursts in demand.

During the event, the system sustained 23.8 Tbps peak bandwidth, 1.7 million QPS in node recommendation, and up to 710k concurrent viewers per stream. To evaluate QoE under this workload, we compared RLive with traditional CDNs. Table 4 shows results from the Dec. 4 match, demonstrating that RLive leverages best-effort resources to handle unprecedented demand while maintaining CDN-grade QoE.

7.4 Discussion

Protocol Support. RLive adopts FLV as the primary CDN-to-edge protocol (i.e., between the CDN and best-effort nodes) due to its low latency and widespread deployment in production pull-based streaming. As shown in Section 7.1, RLive achieves QoE on par with CDN-only delivery under FLV. To demonstrate protocol generality, we also prototyped RLive with RTM, a WebRTC-based protocol described in [85]. Figure 13 shows the relative QoE difference between the RTM-only baseline and the RTM+RLive configuration, with RTM-only as the control group. A/B tests reveal a marginal increase (~1%) in E2E latency, while bitrate and rebuffering remain nearly unchanged. Notably, RTM+RLive supports the same user load with less CDN bandwidth by leveraging best-effort edge resources. These results validate that RLive generalizes effectively across CDN-to-edge protocols.

Fallback Threshold Tradeoff. To absorb packet reordering, RLive uses a client-side playback buffer with a fallback threshold: when the buffer occupancy drops below this threshold, the client switches to CDN full-stream delivery. We evaluate different threshold settings and find that lowering from 500 ms to 400 ms causes only minor rebuffering overhead, while further reducing to 300 ms leads to a sharp QoE degradation. We therefore adopt a 400 ms threshold in production to balance latency and playback smoothness.

8 Discussion

8.1 Deployment Experiences

During the actual deployment of RLive, we have accumulated a series of valuable experiences.

Fine-Grained NAT Classification for Improved Traversal. Successful NAT traversal is crucial for effective best-effort resource use. To prevent overload on public-IP nodes or easily punched NATs, we target hard NAT types like Port-Restricted and Symmetric NATs. Beyond the RFC 5780 classification [44], our deployment uncovered two new behaviors: (*i*) incremental port mappings, and (*ii*) sequential firewall filtering. Building on these insights, we refine NAT types and develop targeted techniques such as port prediction and asymmetric TTL tuning to improve traversal rates. These optimizations expand the usable node pool by approximately 22%, and we believe they offer broader benefits for enhancing P2P connectivity in heterogeneous environments.

Accelerating Frame Recovery via DNS Bypass. To reduce recovery latency, best-effort nodes embed the publisher's IP in regular data packets, enabling clients to bypass DNS lookups and establish direct connections. When losses persist, clients may switch to pulling streams from dedicated nodes. In both cases, IP propagation speeds up redirection and recovery.

Handling Heterogeneous Bottlenecks with Quota-Based Availability. We find advertised bandwidth of heterogeneous best-effort nodes is far less reliable than dedicated servers. Moreover, bandwidth is not always the bottleneck; nodes can encounter limits in CPU, memory, or other resources even with low bandwidth utilization (~10%). This heterogeneity complicates capacity planning and resource allocation. To address this, each node logs its bottleneck locally during stress testing and runtime monitoring. We then evaluate node availability using a quota-based approach, which considers each node's specific bottleneck and multi-dimensional constraints, rather than relying solely on bandwidth utilization.

When Optimality Hurts Scalability. While end-user mapping optimization can theoretically improve scalability and user experience, our deployment experience in hyperscale, heterogeneous environments tells a different story. In practice, aggressively pursuing precision yields only marginal gains but introduces instability—stemming from complex optimization logic and imperfect inputs such as delayed updates, uneven observability, and aggregation errors. We find that favoring a more responsive, resilient strategy over exhaustive optimization leads to a more stable and scalable system in the real world.

Applicability to VoD. While our system is designed for live streaming, many of its components generalize naturally to VoD scenarios. In particular, the collaborative control plane can be directly reused for source selection to improve throughput. Moreover, multi-source delivery is even simpler in VoD, as the content is pre-generated and can be pre-sliced, eliminating the need for runtime stream reassembly.

8.2 Lessons Learned

Our approach supports broader edge-based solutions that offload both data delivery and computation to edge nodes while preserving stringent QoE guarantees. We summarize our lessons learned below.

Multi-layer feedback for fine-grained control. The global scheduler initially relied on aggregated statistics from edge nodes to reduce control overhead. However, this aggregation cannot capture local performance fluctuations in real time, often resulting in overestimating the quality of poor-performing nodes. To address this, we adopt two strategies: global explore-exploit balancing and edge-driven lightweight feedback. At the global level, the scheduler mixes historically good nodes (exploit) with idle or underused candidates (explore) to avoid overloading inaccurate nodes. At the edge, we shift more control and failure detection responsibilities via simple heuristics like locally blacklisting persistently failing nodes. This combination of global exploration and localized feedback has proven robust and practical in dynamic environments.

Distributed transmission control for robustness. As the system scales, centralized control channels become overloaded and fragile. In our early design, we followed the common practice of simplifying ordering logic by computing frame-ordering metadata at a centralized server and transmitting it separately from the data payload. However, failures in the centralized metadata channel prevent clients from reconstructing the stream, leading to significant playback delays. We thus embed the contextual metadata directly into data packets, which are distributed alongside the payload by best-effort nodes. This allows clients to reconstruct streams independently, without relying on a centralized metadata path, significantly improving resilience. Our experience points to a broader principle for hyperscale edge-assisted systems, where centralized channels often become bottlenecks: embedding contextual control information into the data path enhances robustness by eliminating single points of failure.

QoE-Driven Aggressiveness with Controlled Fallback. Our system prioritizes user experience and adopts proactive fallback strategies guided by real-time QoE metrics. We aggressively utilize the dedicated CDN to deliver the initial GoP for fast startup, recover lost frames, and support playback when the client buffer level is low. These strategies provide a controlled and reliable way to safeguard QoE, even when unstable edge resources are involved, at the cost of occasional redundancy (*e.g.*, duplicate packets from both CDN and best-effort nodes). To balance performance and efficiency, we also explore ways to reduce unnecessary redundancy[23].

8.3 Open Questions

Several open directions remain. First, the streaming latency can be further reduced by detecting causes of delayed or missing frames, such as link degradation or edge-side processing delays. Once identified, the system could react in real time by adjusting source selection, retransmission, or scheduling, thereby improving playback continuity. Second, congestion control algorithms (CCAs) for multi-source multi-substream relays will enhance transmission between best-effort nodes and clients. Third, RLive currently applies static substream partitioning via frame-level hashing, which is simple but lacks flexibility for variable frame size or importance. Adaptive scheduling could direct critical or large frames to more stable nodes, reducing tail latency. Finally, exploring RLive's applicability to other latency- and QoE-sensitive workloads, like cloud gaming and 360-degree video, is also worth further investigation.

9 Related work

Live streaming optimization. Crowd-sourced live streaming (CSLV) development has driven decade-long optimization efforts to achieve millisecond latency and high bandwidth, addressed through inject-side [33, 61, 80] and distribution-side [8, 10, 29, 30, 45, 48, 63, 71, 81] enhancements. Growing live traffic has spurred bandwidth cost-saving strategies: P2P-CDN hybrid architectures [68, 73, 74, 76–78] and underutilized resource exploitation [62, 67, 83], alongside dedicated optimization frameworks [11, 65, 84]. In contrast, RLive's goal is to take full advantage of cheaper best-effort nodes to scale existing live CDNs with robust and competitive QoE.

Multi-source transmission. Recent multipath schemes (MPTCP [19, 25, 58, 79], MPRTP [50, 60], MPQUIC [16, 89]) adopt one-source-to-client patterns to enhance QoE. Unlike them, RLive proposes a multi-source-to-client architecture for robustness. Existing multi-source works [9, 15, 78] rely on content redundancy, while we propose no-redundancy transmission with QoE-driven loss recovery. Some recent no-redundancy studies [3, 34, 59, 64, 66, 67, 83, 86] target VoD via receiver-driven request-push mode. RLive's redundancy-free transmission differs from these approaches and innovates with a publisher-driven subscribe-push paradigm, specifically optimized for live streaming's ultra-low latency constraints and ultra-high QoE demands.

Hybrid control plane. Centralized traffic engineering (TE) controllers [1, 5, 12, 20, 29, 30, 38, 42, 43, 49, 53, 69] typically operate on large timescales (minutes/hours) for time-intensive optimizations. Distributed approaches [18, 22, 31, 47] enable local decision-making to minimize cross-domain data transfer and control latency. Hybrid architectures [48, 70] combine centralized optimization with decentralized agility but assume stable and limited infrastructure. In contrast, RLive's collaborative control plane addresses the instability and hyper-scale of edge controllers, ensuring robust hybrid control.

Peer-assisted content distribution. Early work [3, 34, 86] applied peer-assisted designs to non-real-time content distribution, focusing on locality-aware peer selection for ISP cost reduction [4, 7, 13, 32] and network coding for block propagation [21, 37]. While both peer-assisted systems and RLive leverage idle edge resources, early approaches were cost-driven with limited concern for QoE, whereas RLive is explicitly QoE-driven to deliver CDN-grade performance.

10 Conclusion

This paper presents RLive, a robust and scalable live streaming delivery system. RLive introduces a multi-layer collaborative control plane and a redundancy-free multi-source data plane, aiming to scale current CDNs through best-effort edge nodes while ensuring robustness. Successfully deployed in ByteDance CDN for over 3 years, RLive yields significant improvements in QoE, which demonstrates the transformative potential of multi-source streaming, paving the way for more resilient and cost-effective live content delivery.

11 Acknowledgement

We sincerely thank our shepherd, Saikat Guha, and the anonymous reviewers for their valuable comments. This work is supported by National Key R&D Program of China (Grant No. 2022YFB3103000).

References

[1] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. 2021. Contracting wide-area network topologies to solve flow problems quickly. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 175–200.

[2] Micah Adler, Ramesh K Sitaraman, and Harish Venkataramani. 2011. Algorithms for optimizing the bandwidth cost of content delivery. *Computer Networks* 55, 18 (2011), 4007–4020.

[3] Vikash Agarwal and Reza Rejaie. 2005. Adaptive multisource streaming in heterogeneous peer-to-peer networks. In *Multimedia Computing and Networking 2005*, Vol. 5680. SPIE, 13–25.

[4] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. 2007. Can ISPs and P2P users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review* 37, 3 (2007), 29–40.

[5] Abd AlRhman AlQiam, Yuanjun Yao, Zhaodong Wang, Satyajeet Singh Ahuja, Ying Zhang, Sanjay G Rao, Bruno Ribeiro, and Mohit Tawarmalani. 2024. Transferable Neural WAN TE for Changing Topologies. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 86–102.

[6] Congkai An, Huanhuan Zhang, Shibo Wang, Jingyang Kang, Anfu Zhou, Liang Liu, Huadong Ma, Zili Meng, Delei Ma, Yusheng Dong, et al. 2025. Tooth: Toward Optimal Balance of Video QoE and Redundancy Cost by Fine-Grained FEC in Cloud Gaming Streaming. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 635–651.

[7] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang. 2006. Improving traffic locality in BitTorrent via biased neighbor selection. In *26th IEEE international conference on distributed computing systems (ICDCS'06)*. IEEE, 66–66.

[8] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. 2018. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn. *ACM SIGCOMM Computer Communication Review* 48, 1 (2018), 28–34.

[9] Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Negru, Jordi Mongay Batalla, and Eugen Borcoci. 2017. QOE enhancement through cost-effective adaptation decision process for multiple-server streaming over HTTP. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1–6.

[10] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. 2018. Odin: Microsoft's scalable Fault-Tolerant CDN measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 501–517.

[11] Huan Chen, Huiyou Zhan, Haisheng Tan, Huang Xu, Weihua Shan, Shiteng Chen, and Xiang-Yang Li. 2022. Online Traffic Allocation Based on Percentile Charging for Practical CDNs. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[12] Xiang Chen, Qingjiang Xiao, Hongyan Liu, Qun Huang, Dong Zhang, Xuan Liu, Longbing Hu, Haifeng Zhou, Chunming Wu, and Kui Ren. 2024. Eagle: Toward Scalable and Near-Optimal Network-Wide Sketch Deployment in Network Measurement. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 291–310.

[13] David R Choffnes and Fabián E Bustamante. 2008. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. *ACM SIGCOMM Computer Communication Review* 38, 4 (2008), 363–374.

[14] JD Cloud. 2024. *JD Cloud Services*. https://m-console-buy.jdcloud.com/init?product=Z

[15] Simon Da Silva, Joachim Bruneau-Queyreix, Mathias Lacaud, Daniel Négru, and Laurent Réveillère. 2018. MUSLIN: Achieving high, fairly shared QoE through multi-source live streaming. In *Proceedings of the 23rd Packet Video Workshop*. 54–59.

[16] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath quic: Design and evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*. 160–166.

[17] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review* 41, 4 (2011), 362–373.

[18] Anwar Elwalid, Cheng Jin, Steven Low, and Indra Widjaja. 2001. MATE: MPLS adaptive traffic engineering. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, Vol. 3. IEEE, 1300–1309.

[19] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. 2016. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *2016 IFIP networking conference (IFIP networking) and workshops*. IEEE, 431–439.

[20] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. 2015. C3: Internet-scale control plane for video quality optimization. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 131–144.

[21] Christos Gkantsidis and Pablo Rodriguez Rodriguez. 2005. Network coding for large scale content distribution. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, Vol. 4. IEEE, 2235–2245.

[22] Fei Gui, Songtao Wang, Dan Li, Li Chen, Kaihui Gao, Congcong Min, and Yi Wang. 2024. RedTE: Mitigating subsecond traffic bursts with real-time and distributed traffic engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 71–85.

[23] Peisheng Guo, Jiao Zhang, Yuqing Wang, Haozhe Li, Zhichen Xue, Yajie Peng, Rui Han, Xiaofei Pang, Tao Huang, Ruili Fang, et al. 2025. HELDR: Packet Loss Detection and Retransmission for Live Streaming Hyper-Edge Network. In *2025 IEEE/ACM 33rd International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[24] Syed Hasan, Sergey Gorinsky, Constantine Dovrolis, and Ramesh K Sitaraman. 2014. Trade-offs in optimizing the cache deployments of CDNs. In *IEEE INFOCOM 2014-IEEE conference on computer communications*. IEEE, 460–468.

[25] Per Hurtig, Karl-Johan Grinnemo, Anna Brunstrom, Simone Ferlin, Özgü Alay, and Nicolas Kuhn. 2018. Low-latency scheduling in MPTCP. *IEEE/ACM transactions on networking* 27, 1 (2018), 302–315.

[26] iCDN. 2024. *iCDN Prime, Hybrid CDN-node as a service*. https://www.icdntele.com/index.php/icdn-prime

[27] Adobe Systems Incorporated. 2010. *Adobe Flash Video File Format Specification*. https://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf

[28] Satoshi Itaya, Tomoya Enokido, and Makoto Takizawa. 2005. A high-performance multimedia streaming model on multi-source streaming approach in peer-to-peer networks. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, Vol. 1. IEEE, 27–32.

[29] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. 2016. CFA: A practical prediction system for video QoE optimization. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 137–150.

[30] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*. 393–406.

[31] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. 2005. Walking the tightrope: Responsive yet stable traffic engineering. *ACM SIGCOMM Computer Communication Review* 35, 4 (2005), 253–264.

[32] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. 2005. Should internet service providers fear peer-assisted content distribution?. In *Proceedings of the 5th ACM SIGCOMM Conference on*

Internet Measurement. 6–6.

[33] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 107–125.

[34] Greg Kowalski and Mohamed Hefeeda. 2009. Empirical analysis of multi-sender segment transmission algorithms in peer-to-peer streaming. In *2009 11th IEEE International Symposium on Multimedia*. IEEE, 243–250.

[35] Federico Larumbe and Abhishek Mathur. 2015. Under the hood: Broadcasting live video to millions. https://engineering.fb.com/2015/12/03/ios/under-the-hood-broadcasting-live-video-to-millions/.

[36] HyunJong Lee, Jason Flinn, and Basavaraj Tonshal. 2018. RAVEN: Improving interactive latency for the connected car. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 557–572.

[37] Uichin Lee, Joon-Sang Park, Joseph Yeh, Giovanni Pau, and Mario Gerla. 2006. Code torrent: content distribution using network coding in vanet. In *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*. 1–5.

[38] Geng Li, Shuihai Hu, and Kun Tan. 2024. Panorama: Optimizing Internet-scale Users' Routes from End to End. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 935–949.

[39] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, et al. 2022. Livenet: a low-latency video transport network for large-scale live streaming. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 812–825.

[40] Zhenyu Li, Jiali Lin, Marc-Ismael Akodjenou, Gaogang Xie, Mohamed Ali Kaafar, Yun Jin, and Gang Peng. 2012. Watching videos from everywhere: a study of the pptv mobile vod system. In *Proceedings of the 2012 internet measurement conference*. 185–198.

[41] Guanfeng Liang and Ulaş C Kozat. 2013. Fast cloud: Pushing the envelope on delay performance of cloud storage with coding. *IEEE/ACM Transactions on Networking* 22, 6 (2013), 2012–2025.

[42] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. 2012. A case for a coordinated internet video control plane. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 359–370.

[43] Ximeng Liu, Shizhen Zhao, Yong Cui, and Xinbing Wang. 2024. FIGRET: Fine-Grained Robustness-Enhanced Traffic Engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 117–135.

[44] Derek MacDonald and Bruce Lowekamp. 2010. NAT behavior discovery using session traversal utilities for NAT (STUN). RFC 5780. https://www.rfc-editor.org/info/rfc5780

[45] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*. 197–210.

[46] Zili Meng, Xiao Kong, Jing Chen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. 2024. Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 907–926.

[47] Nithin Michael and Ao Tang. 2014. Halo: Hop-by-hop adaptive link-state optimal routing. *IEEE/ACM Transactions on Networking* 23, 6 (2014), 1862–1875.

[48] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. 2015. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 311–324. doi:10.1145/2829988.2787475

[49] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 521–537.

[50] Ekaterina Pakulova, Konstantin Miller, and Adam Wolisz. 2017. Adaptive low-delay video streaming in heterogeneous wireless networks using MPRTP. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 14–19.

[51] Roger Pantos and William May. 2017. *HTTP live streaming*. Technical Report.

[52] KyoungSoo Park and Vivek S Pai. 2006. Scale and Performance in the CoBlitz Large-File Distribution Service.. In *NSDI*.

[53] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. 2023. DOTE: Rethinking (Predictive) WAN Traffic Engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1557–1581.

[54] Qiniu. 2024. *Qiniu CDN Pricing*. https://www.qiniu.com/prices/qcdn

[55] DataHorizzon Research. 2024. Livestream Commerce Market To Reach USD 3,058.8 Billion By 2032, Report By DataHorizzon Research. https://www.globenewswire.com/news-release/2024/04/19/2866112/0/en/Livestream-Commerce-Market-To-Reach-USD-3-058-8-Billion-By-2032-Report-By-DataHorizzon-Research.html. Accessed: 2024-05-16.

[56] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. 2023. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 953–971.

[57] Amazon Web Services. 2025. *Video Latency in Live Streaming*. https://aws.amazon.com/media/tech/video-latency-in-live-streaming

[58] Hang Shi, Yong Cui, Xin Wang, Yuming Hu, Minglong Dai, Fanzhao Wang, and Kai Zheng. 2018. STMS: Improving MPTCP throughput under heterogeneous networks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 719–730.

[59] Pengbo Si, F Richard Yu, Hong Ji, and Victor CM Leung. 2009. Distributed multisource transmission in wireless mobile peer-to-peer networks: a restless-bandit approach. *IEEE Transactions on Vehicular Technology* 59, 1 (2009), 420–430.

[60] Varun Singh, Saba Ahsan, and Jörg Ott. 2013. MPRTP: multipath considerations for real-time media. In *Proceedings of the 4th ACM Multimedia Systems Conference*. 190–201.

[61] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2020. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM transactions on networking* 28, 4 (2020), 1698–1711.

[62] Yu Tian, Zhenyu Li, Matthew Yang Liu, Jian Mao, Gareth Tyson, and Gaogang Xie. 2024. Cost-Saving Streaming: Unlocking the Potential of Alternative Edge Node Resources. In *Proceedings of the 2024 ACM on Internet Measurement Conference*. 580–587.

[63] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, Maurizio M Munafo, and Sanjay Rao. 2011. Dissecting video server selection strategies in the YouTube CDN. In *2011 31st International Conference on Distributed Computing Systems*. IEEE, 248–257.

[64] Haiping Wang, Ruixiao Zhang, Chaojun Li, Zhichen Xue, Yajie Peng, Xiaofei Pang, Yixuan Zhang, Shaorui Ren, and Shu Shi. 2024. Twist: A Multi-site Transmission Solution for On-demand Video Streaming. *Proceedings of the ACM on Networking* 2, CoNEXT2 (2024), 1–19.

[65] Yizong Wang, Dong Zhao, Chenghao Huang, Fuyu Yang, Teng Gao, Anfu Zhou, Huanhuan Zhang, Huadong Ma, Yang Du, and Aiyun Chen. 2023. TrafAda: Cost-aware traffic adaptation for maximizing bitrates in live streaming. *IEEE/ACM Transactions on Networking* 32, 1 (2023), 96–109.

[66] Dehui Wei, Jiao Zhang, Haozhe Li, Zhichen Xue, Yajie Peng, Xiaofei Pang, Rui Han, Yan Ma, and Jialin Li. 2024. Swarm: Cost-Efficient Video Content Distribution with a Peer-to-Peer System. arXiv:2401.15839 [cs.NI]

[67] Dehui Wei, Jiao Zhang, Haozhe Li, Zhichen Xue, Yajie Peng, Xiaofei Pang, Yuanjie Liu, Rui Han, and Jialin Li. 2024. Pscheduler: QoE-Enhanced MultiPath Scheduler for Video Services in Large-scale Peer-to-Peer CDNs. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2508–2517.

[68] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, and Heung-Keung Chai. 2006. Analysis of a CDN–P2P hybrid architecture for cost-effective streaming media distribution. *Multimedia Systems* 11 (2006), 383–399.

[69] Zhiying Xu, Francis Y Yan, Rachee Singh, Justin T Chiu, Alexander M Rush, and Minlan Yu. 2023. Teal: Learning-accelerated optimization of wan traffic engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 378–393.

[70] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. 2017. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 432–445.

[71] Ziwen Ye, Qing Li, Chunyu Qiao, Xiaoteng Ma, Yong Jiang, Qian Ma, Shengbin Meng, Zhenhui Yuan, and Zili Meng. 2024. KEPC-Push: A Knowledge-Enhanced Proactive Content Push Strategy for Edge-Assisted Video Feed Streaming. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 321–338.

[72] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2022. Neuroscaler: Neural video enhancement at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 795–811.

[73] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. 2009. 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*. 25–34.

[74] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. 2010. Livesky: Enhancing cdn with p2p. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 6, 3 (2010), 1–19.

[75] Minlan Yu, Wenjie Jiang, Haoyuan Li, and Ion Stoica. 2012. Tradeoffs in CDN designs for throughput oriented traffic. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 145–156.

[76] Ge Zhang, Wei Liu, Xiaojun Hei, and Wenqing Cheng. 2014. Unreeling Xunlei Kankan: Understanding hybrid CDN-P2P video-on-demand streaming. *IEEE Transactions on Multimedia* 17, 2 (2014), 229–242.

[77] Huanhuan Zhang, Congkai An, Anfu Zhou, Chaoyue Li, Xi Liu, Jialiang Pei, Yifan Zhu, Liang Liu, and Huadong Ma. 2024. Reviving Peer-to-Peer Networking for Scalable Crowdsourced Live Video Streaming. *IEEE/ACM Transactions on Networking* (2024).

[78] Huanhuan Zhang, Congkai An, Anfu Zhou, Yifan Zhu, Weilin Sun, Yixuan Lu, Jiahao Chen, Liang Liu, Huadong Ma, and Aiguo Fei. 2024. Venus: Enhancing QoE of Crowdsourced Live Video Streaming by Exploiting Multiflow Viewer Assistance. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 170–184.

[79] Han Zhang, Wenzhong Li, Shaohua Gao, Xiaoliang Wang, and Baoliu Ye. 2019. ReLeS: A neural adaptive multipath scheduler based on deep reinforcement learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1648–1656.

[80] Huanhuan Zhang, Anfu Zhou, and Huadong Ma. 2022. Improving mobile interactive video QoE via two-level online cooperative learning. *IEEE Transactions on Mobile Computing* 22, 10 (2022), 5900–5917.

[81] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Hanyu Li, Jiangchuan Liu, and Lifeng Sun. 2020. Leveraging QoE heterogenity for large-scale livecaset scheduling. In *Proceedings of the 28th ACM International Conference on Multimedia*. 3678–3686.

[82] Rui-Xiao Zhang, Ming Ma, Tianchi Huang, Haitian Pang, Xin Yao, Chenglei Wu, Jiangchuan Liu, and Lifeng Sun. 2019. Livesmart: A qos-guaranteed cost-minimum framework of viewer scheduling for crowdsourced live streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*. 420–428.

[83] Rui-Xiao Zhang, Haiping Wang, Shu Shi, Xiaofei Pang, Yajie Peng, Zhichen Xue, and Jiangchuan Liu. 2024. Enhancing Resource Management of the World's Largest PCDN System for On-Demand Video Streaming. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 951–965.

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

[85] Wei Zhang, Tong Meng, Xianhua Zeng, Wei Yang, Changqing Yan, Chao Li, Chenguang Li, Feng Qian, Junfeng Yang, Lei Zhang, et al. 2025. Harnessing WebRTC for Large-Scale Live Streaming. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 1199–1212.

[86] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Y-SP Yum. 2005. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, Vol. 3. IEEE, 2102–2111.

[87] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. 2021. SENSEI: Aligning video streaming quality with dynamic user sensitivity. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 303–320.

[88] Yuankang Zhao, Furong Yang, Gerui Lv, Qinghua Wu, Yanmei Liu, Jiuhai Zhang, Yutang Peng, Feng Peng, Hongyu Guo, Ying Chen, et al. 2025. {MARC}:{Motion-Aware} Rate Control for Mobile E-commerce Cloud Rendering. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 217–232.

[89] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. 2021. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 418–432.