USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

HCDN: Coordinated Stream Scheduling for Cost-Effective Live Video Delivery

Liying Wang, *Peking University;* Jing Liu, *ByteDance;* Yuhan Zhou and Chengke Wang, *Peking University;* Mingming Lu, Qingyue Li, Song Geng, Linsen Wang, Kaida Hu, Haoyuan Huang, Shimao Tian, Ri Lu, and Mingfei Hao, *ByteDance;* Chenren Xu, *Peking University; and Key Laboratory of High Confidence Software Technologies, Ministry of Education (PKU);* Shu Shi, *ByteDance*

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

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

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

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

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

HCDN: Coordinated Stream Scheduling for Cost-Effective Live Video Delivery

Liying Wang[ℙ], Jing Liu[𝔹✉*], Yuhan Zhou[ℙ], Chengke Wang[ℙ], Mingming Lu[𝔹], Qingyue Li[𝔹]
Song Geng[𝔹], Linsen Wang[𝔹], Kaida Hu[𝔹], Haoyuan Huang[𝔹], Shimao Tian[𝔹], Ri Lu[𝔹]
Mingfei Hao[𝔹], Chenren Xu[ℙ𝕂✉*], Shu Shi[𝔹]
[ℙ]*School of Computer Science, Peking University* [𝔹]*ByteDance*
[𝕂]*Key Laboratory of High Confidence Software Technologies, Ministry of Education (PKU)*

Abstract – Live video streaming is a major source of today's Internet traffic, yet its delivery through CDNs incurs massive bandwidth costs from both CDN edge traffic (to users) and internal traffic (between CDN nodes). To understand these costs, we conduct a large-scale measurement study of ByteDance's global in-production live CDN. It reveals two opportunities to reduce CDN bandwidth cost: *(i)* lowering average edge price by incorporating cost-effective best-effort infrastructure into the CDN edge, and *(ii)* improving CDN tree efficiency via finer-grained stream scheduling, accounting for detailed stream characteristics such as nonuniform popularity, heterogeneous stream formats, which amplify internal traffic on cache misses in our prior CDN. To exploit these opportunities into a practical system while addressing challenges in scheduling strategy management and user quality of experience, we design, implement, and deploy HCDN. It comprises *(i)* an augmented CDN edge with cost-effective *best-effort nodes* and *multihomed nodes*, *(ii)* the OpenTiga scheduling framework, which coordinates strategies through modular orchestration and proactive client-side Quality-of-Experience (QoE) assurance, and *(iii)* a set of redirection-based stream scheduling strategies. Over four years of incremental deployment across more than 500 edge clusters, HCDN reduces relative bandwidth cost by 36% while incurring modest overhead and preserving QoE. We further report the deployment experience of HCDN.

1 Introduction

Live video streaming constitutes a major share of today's Internet traffic, with platforms serving billions of users [1–8]. Under this scale, these platforms rely on content delivery networks (CDNs) to deliver the live videos. CDNs organize servers into geographically distributed *nodes* (*i.e.,* clusters) to cache and deliver live streams [9–11], which incur over $100 million annually in network bandwidth costs [12].

In live streaming CDNs, video delivery is organized around distribution topologies (CDN trees) [9–11]: before reaching viewers, these live streams are transported through intermediate CDN nodes. This motivates us to investigate bandwidth cost through the following three key factors: *(i) Edge price*, determined by the cost of delivering a byte to viewers from the CDN edge (*i.e.,* leaf nodes in the CDN tree structure); *(ii) Tree efficiency*, which measures how much CDN internal traffic is induced per unit of CDN edge bandwidth; and *(iii) Internal price*, determined by the cost of delivering a byte to the CDN edge from the video source. To this end, we conduct a large-scale measurement study in ByteDance's global CDN, and our analysis uncovers the following two key opportunities to lower the relative bandwidth cost, focusing on edge price (§2.3) and tree efficiency (§2.5).

- The average edge price can be reduced by integrating cost-effective best-effort edge nodes into the architecture, which has been proven effective in traditional (*i.e.,* not designed for live video streaming) CDNs [11,13–15].
- The tree efficiency can be reduced by finer-grained stream aggregation strategies. To reduce internal traffic, prior work aggregates streams onto fewer edge nodes based on global hotness (total online viewers) of each stream [16]. We show that aggregation must account for finer-grained characteristics than global hotness alone. *(i)* Hot streams can still exhibit low tree efficiency at nodes where they are locally cold due to nonuniform viewer geography. *(ii) Frozen streams* (*i.e.,* extremely cold streams) are too sparse across regions and ISPs for aggregation to be effective [16], motivating nodes that support cross-ISP or cross-region delivery [17–19]. *(iii)* Substreams and patch streams are common in live CDNs [14, 20, 21], constitute $\sim$ 30% of streams in our CDN, and incur 1.72 $\sim$ 5.76$\times$ higher internal traffic efficiency than full streams because a cache miss still triggers an upstream fetch of the full stream.

These findings suggest a guideline we can follow to reduce bandwidth cost: *augmenting the CDN edge with new node types, such as cost-effective best-effort nodes and cross-ISP or cross-region-capable nodes, and scheduling streams to map requests to suitable nodes with fine-grained strategies.* However, realizing this guideline into practical system raises the following two key challenges.

- **QoE Concerns.** While DNS-based scheduling [22] and anycast [23, 24] are too coarse-grained and slow to support stream-level scheduling, HTTP redirection enables

*Corresponding authors.
✉: liujing.bbd@bytedance.com ✉: chenren@pku.edu.cn

fine-grained, fast-converging control [16, 25, 26], where nodes redirect matching requests and clients retry at the target. However, redirection adds an extra RTT and connection setup overhead, which Akamai describes as acceptable only for large downloads [22]. Our experiment in §3.2.1 confirms that such redirections decreases QoE in terms of startup delay and playback stalls.

- **Long-Term Strategy Management.** Over 4 years of operation, we integrate new node types and identify new optimization opportunities (*e.g.,* non-uniform stream hotness), which requires continuously adding new strategies and revising existing ones. At production scale, uncoordinated ad-hoc patches can create strategy conflicts and unmaintainable technical debt, and strategy development often requires production code changes and cross-team coordination. Moreover, fine-grained scheduling decisions, if not carefully designed, can impose substantial control-plane overhead in edge memory, central memory, and control bandwidth.

We present HCDN, our in-production hybrid CDN which fully exploit the opportunities while addressing the practical challenges. The system design comprises three components. First, we augment our prior CDN architecture with two new types of edge nodes: *multihomed edge nodes*, which serve frozen streams that sparsely span ISP-region partitions, and *alternative edge nodes*, which leverage cost-effective best-effort infrastructure to lower the average edge price. Second, we design the OpenTiga scheduling framework to address the two practical challenges and enable coordinated stream scheduling at scale. To avoid user QoE degradation, it leverages client-side assistance [27, 28] to achieve *Proactive QoE Assurance*, which identifies upcoming streams directly from the recommendation system and proactively performs redirection, connection establishment, and stream pulling in advance. To perform long-term strategy management, it acts as a central orchestrator that manages multiple strategies. It enables modular development and lightweight coordination by decoupling rapidly evolving strategies from stable mechanisms. When new opportunities are discovered, the orchestrator allows developers to safely add or tune strategies without destabilizing the core system. Third, we apply four scheduling strategies that exploit opportunities to reduce bandwidth cost: *(i)* offloading frozen streams to multihomed edge nodes, *(ii)* aggregating cold streams onto a small subset of regular edge nodes, *(iii)* migrating hot streams from cold regular nodes to hot ones, and *(iv)* offloading hot streams from hot regular nodes to alternative edge nodes.

Over the past four years, we have migrated from the prior CDN architecture to HCDN, now operating across more than 500 production edge clusters. This large-scale deployment demonstrates that HCDN reduces relative bandwidth cost by 36%, with modest overhead ($<$ 270 MB controller memory and $<$ 150 Mbps control bandwidth) for production scale, while maintaining user QoE through proactive quality assurance. We also demonstrate the impact of individual strategies along the incremental deployment timeline. Moreover, we report our experience from the deployment.

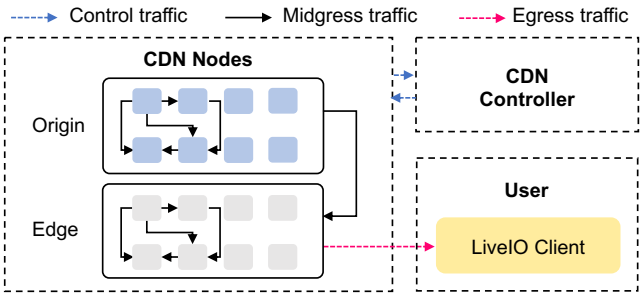


Figure 1: Our prior CDN architecture.

Contributions.

- We identify opportunities to reduce bandwidth cost based on measurement and analysis of our large-scale in-production live video streaming CDN (§2).
- We design, implement, and deploy HCDN, which combines augmented CDN infrastructure, a scheduling framework, and a set of stream scheduling strategies to minimize CDN bandwidth cost (§3∼4).
- We demonstrate that HCDN reduces relative bandwidth cost with modest overhead while preserving user QoE through our production deployment (§5), and share our lessons learned from deployment (§6).

Ethical claims. All data statistics used in this work were collected with explicit consent and anonymized to protect privacy. This study raises no ethical concerns.

2 Background and Motivation

2.1 Our Prior CDN

Our prior live streaming CDN (Fig. 1) serves *streams* (*i.e.,* content-level channels such as a concert broadcast or a gaming livestream) on multiple CDN nodes. On each node, users access a stream via individual *sessions*, each corresponding to one user pulling the stream. Users request streams through the LiveIO client: the recommendation system provides a stream ID, the client resolves it via DNS to the default node IP, then connects to the node to receive the stream.

CDN Nodes. The CDN nodes are divided into two main types: *(i) origin nodes* (purple in Fig. 1) that connect to content providers to ingest live video uploads, and *(ii) edge nodes* (gray in Fig. 1) that connect to users to directly serve their requests. Some origin nodes are organized in a flat circular topology (purple nodes connected by black arrows in Fig. 1), similar to recent industry designs [9], where nodes simultaneously serve downstream requests and relay traffic. Other origin nodes operate in a standalone mode. This structure also applies to edge nodes: some form a flat circular topology, while others remain isolated.

CDN Traffic. When a client requests a live stream, the edge

node serves it directly if available locally. This delivery from CDN edge nodes to users constitutes *egress traffic* (also referred to as *edge traffic*). Otherwise, if the node belongs to a circular topology, it forwards the request to peers in the same group. If no peer can provide the stream, or if the node is standalone, the edge initiates *midgress* to fetch the stream from the origin. This internal data transmission between CDN nodes (including origin-to-edge and inter-edge relay) constitutes *midgress traffic*.

CDN Controller. The controller provides basic control-plane functionality. It periodically pushes configuration updates to origin and edge nodes, detects node failures, and performs load balancing across peer groups.

2.2 Understanding CDN Bandwidth Cost

In a typical live streaming CDN, the total bandwidth cost C is the sum of the egress cost C_E and the midgress cost C_{Mid}:

$$C = C_E + C_{Mid} = \overline{P_E} \cdot BW_E \cdot r_E^{95} + \overline{P_{Mid}} \cdot BW_{Mid} \cdot r_{Mid}^{95} \quad (1)$$

where BW_E and BW_{Mid} denote the total egress and midgress bandwidth. Because CDN bandwidth is billed by peak usage (P95) [29, 30], the equation includes r_E^{95} and r_{Mid}^{95}, which represent the *95th-percentile utilization factors* for egress and midgress bandwidth. And $\overline{P_E} = C_E/BW_E$ is the average egress price, and $\overline{P_{Mid}} = C_{Mid}/BW_{Mid}$ is the average midgress price. We focus on the cost effectiveness of CDN bandwidth, captured by the relative bandwidth cost $\overline{P}$, defined as the ratio of the total cost to the egress bandwidth:

$$\overline{P} = \frac{C}{BW_E} = \overline{P_E} \cdot r_E^{95} + \overline{P_{Mid}} \cdot \text{MER} \cdot r_{Mid}^{95} \quad (2)$$

where MER is the overall midgress-egress ratio, defined as $\text{MER} = BW_{Mid}/BW_E$. As Eqn. 2 shows, CDN bandwidth cost-effectiveness depends on three key factors: *(i)* the average egress price $\overline{P_E}$, *(ii)* the average midgress price $\overline{P_{Mid}}$, and *(iii)* the midgress-egress ratio (MER). These three metrics map directly to the conceptual factors introduced in §1: $\overline{P_E}$ determines the average *Edge Price*, MER quantifies the *Tree Efficiency*, and $\overline{P_{Mid}}$ represents the average *Internal Price*. In the following sections (§2.3-2.5), we analyze each factor to identify opportunities for reducing the overall relative cost.

2.3 Understanding the Average Egress Price $\overline{P_E}$

There exist different types of edge nodes for CDNs, and the average egress price $\overline{P_E}$ depends on the types of edge nodes they use. A common idea in traditional CDNs to reduce $\overline{P_E}$ is to leverage cost-effective edge resources [13, 14, 31]. To explore similar opportunities in live streaming CDNs, we first examine the available edge node types.

Cost-effective ISP edge nodes can potentially reduce $\overline{P_E}$. In Tab. 1, we compare the available types of edge resources, including multihomed nodes [17–19], regular CDN nodes, ISP edge nodes [31], and P2P CDN (PCDN) nodes [13, 14, 21]. Regular nodes offer high QoE but are expensive; ISP edge nodes are cheaper while still maintaining high QoE; PCDN nodes are the cheapest but degrade QoE. A promising approach is to leverage ISP edge nodes as best-effort resources, thereby reducing the average egress price $\overline{P_E}$.

Challenges. However, the integration of best-effort edge nodes introduces two challenges that must be addressed.

• *Additional Midgress to Best-Effort Nodes.* Best-effort nodes serve as an intermediate layer between users and the existing CDN. When a request is redirected to a best-effort node but the requested stream is not available locally, the node must fetch the stream from an upstream CDN edge node, adding an additional midgress hop to the delivery path. Compared with current practice, where users directly access CDN edges, this extra hop increases midgress traffic and may partially offset the cost benefits of best-effort deployment.

• *Lack of HTTP Compatibility.* The low cost of best-effort resources stems from their limitation of serving only on non-standard (non-80) ports, rather than the default HTTP port supported by browsers, players, and middleboxes. While such non-80 deployment reduces infrastructure and operational cost, it breaks the assumption of HTTP compatibility in the existing live streaming ecosystem. Therefore, supporting these nodes requires explicit client-side operations.

> **Finding 1:** $\overline{P_E}$ can be reduced by leveraging best-effort edge nodes, but this introduces additional midgress and HTTP compatibility issues that must be addressed.

2.4 Understanding the Average Midgress Price $\overline{P_{Mid}}$

The average midgress price $\overline{P_{Mid}}$, defined as C_{Mid}/BW_{Mid}, depends on the unit prices of the midgress data links.

Flat CDN Architecture. Recent work in industry and academia explores flat CDN architectures [9, 32, 33], where CDN nodes form a flat circular topology and interconnect directly. Each node simultaneously acts as *(i)* an edge node serving users, *(ii)* an origin node ingesting streams from providers, and *(iii)* a relay node forwarding streams to peers. This design enables midgress traffic to traverse inter-node links instead of higher-tier nodes. As we observe in our operation, these links exhibit lower unit costs compared to cross-tier links, allowing nodes to fetch streams from peers to effectively lower the average midgress price $\overline{P_{Mid}}$

Our Deployment. In our production system, flat architecture is already in use but deployed incrementally rather than as a full replacement, due to the scale of existing infrastructure. As discussed in §2.1, the current system is therefore hybrid, combining elements of the traditional hierarchical overlay with flat, circular topologies.

Resource Type	Price	QoE	Capacity	Public IP Address	Cross-Partition Capability	Compatibility
Multihomed nodes [17–19]	High	High	Large	✓	✓	Standard (port 80/443 allowed)
Regular nodes	High	High	Large	✓	✗	Standard (port 80/443 allowed)
ISP edge nodes [31]	Medium	High	Small	✓	✗	Restricted (non-HTTP(S) ports only)
PCDN nodes [13, 14, 21]	Low	Low	Small	✗	✗	Restricted: (non-HTTP(S) ports only)

Table 1: Comparison of edge resources

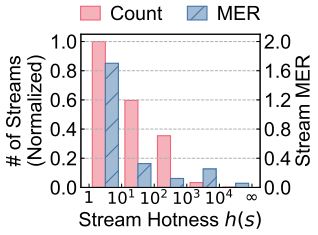


Figure 2: Cold streams have inefficiently high MER.

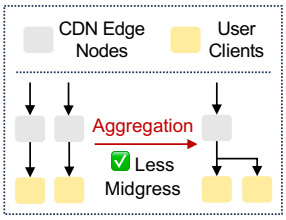


Figure 3: Reducing midgress by stream aggregation.

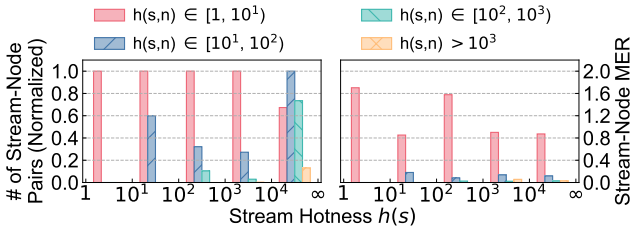


Figure 4: Hot streams are cold on some nodes, where they exhibit high MER.

Remark: Flat CDN architectures already lower $\overline{P_{Mid}}$ and have been adopted in our system, so we focus on the other two factors: $\overline{P_E}$ and MER.

2.5 Understanding the Midgress-Egress Ratio (MER)

MER reflects the efficiency of midgress traffic. A higher MER indicates more cache misses and thus greater bandwidth cost. While a rich body of prior work reduces midgress through cache management in traditional CDNs [34–36] and stream aggregation in live streaming CDNs [16], our analysis of over 1 million production logs reveals overlooked inefficiencies as well as limitations of existing aggregation strategies, driven by the distribution characteristics of live streams.

Analysis Methodology. We analyze MER using global logs from our production CDN. Each record represents a stream s served at a CDN edge node n, reporting statistics such as session counts, midgress bandwidth, and egress bandwidth, as well as metadata such as region, ISP, and request format.

2.5.1 *Cold streams have high MER.*

Cold streams (*i.e.,* streams with very few viewers), generate multiple midgress transfers but little egress traffic, leading to inefficient midgress and inflating MER. We quantitatively confirm this observation in our analysis.

Most streams are cold. To quantify how "hot" a stream is, we define the *stream hotness* of stream s as the number of active sessions pulling s, denoted $h(s)$. As shown in Fig. 2, we group streams into hotness intervals and plot the normalized number of streams in each interval (red bars). The results show that most streams are cold, with over 80% of all streams having hotness $h(s) < 100$.

Cold streams incur inefficient midgress. We analyze midgress efficiency across hotness intervals. In Fig. 2, we plot the average MER (*i.e.,* the ratio of total midgress to total egress bandwidth of all streams in that interval) of each interval with blue bars. The results show that cold streams incur much higher MER than others, with streams of hotness $h(s) < 10$ averaging 1.7 and those with $10 \le h(s) < 100$ averaging 0.32, compared with MER under 0.26 for streams with $h(s) \ge 100$. Since cold streams dominate the stream population, their high MER substantially raises the overall MER and inflates total bandwidth cost.

Aggregating cold streams to reduce MER. A straightforward way to reduce the MER of cold streams is to aggregate them onto a subset of edge nodes [16] (*target nodes*), as illustrated in Fig. 3. New requests are then served only by the target nodes. As existing sessions on other nodes (*source nodes*) end, they stop fetching the stream via midgress, thereby reducing midgress traffic and lowering the MER.

2.5.2 *Hot streams have high MER on "cold nodes".*

Hot streams are cold on some nodes. We further examine hot streams across all nodes that serve them. We define the *stream-node hotness* of stream s on node n, denoted $h(s,n)$, as the number of online sessions pulling s from n. In the left panel of Fig. 4, we group streams by $h(s)$, and within each group of streams, we classify corresponding stream-node pairs by $h(s,n)$. Even hot streams are cold on many nodes, where streams with $1000 \le h(s) < 10000$ and $h(s) > 10000$ have 77% and 27% of nodes with $h(s,n) < 10$, respectively.

Hot streams incur inefficient midgress on cold nodes. We analyze midgress efficiency across intervals of stream-node hotness $h(s,n)$. In the right side of Fig. 4, we plot the average MER of each group of stream-node pairs, grouped first by stream hotness $h(s)$ and then by stream-node hotness $h(s,n)$. Hot streams on cold nodes exhibit high MER, indicating inefficient midgress. For example, when $h(s,n) < 10$, streams

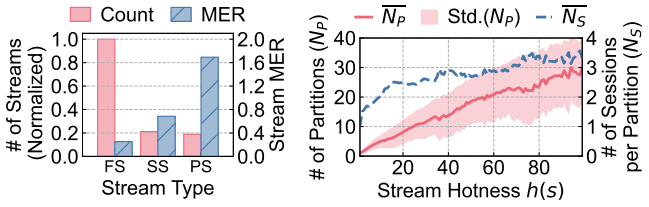


Figure 5: Substreams and patch streams have higher MER than full streams.

Figure 6: Frozen streams span sparsely across partitions, making them hard to aggregate.

with $1000 \leq h(s) < 10000$ and $h(s) \geq 10000$ have average MER values of 0.90 and 0.87, respectively. These values are even higher than the average MER of much colder streams ($10 \leq h(s) < 100$), which is only 0.32 (§2.5.1). Since hot streams are cold on a substantial fraction of nodes, this inefficiency significantly inflates the overall MER.

Hot stream MER requires fine-grained aggregation. The MER of hot streams on cold nodes can also be reduced by aggregation (Fig. 3), which consolidates hot streams onto a smaller subset of nodes. However, for these streams, if hot nodes were selected as aggregation source, they would take a long time to drain, and midgress would persist. Thus, aggregation must proceed from cold nodes toward hot nodes, requiring finer-grained aggregation decisions.

2.5.3 *Substreams and patch streams have high MER.*

Substreams and patch streams account for a considerable proportion. Recently, substreams and patch streams [14, 20, 21], initiated by CDN tenants, are emerging in live video streaming CDNs. A substream is a subset of a full stream for multi-source parallel downloading. A patch stream is a repair flow requesting missing frames to fill substream gaps. In our CDN, full streams (`FS`), substreams (`SS`), and patch streams (`PS`) account for 71%, 15%, and 13% of all streams, respectively, as shown by red bars in Fig. 5.

Substreams and patch streams incur inefficient midgress. We plot the average MER of different stream types in Fig. 5. Full streams show low average MER of 0.25, whereas substreams and patch streams exhibit much higher values (0.68 and 1.69, which are 1.72 and 5.76× higher than full streams, respectively). Substreams are inefficient because a midgress request for one substream fetches the entire stream when the cache is missing or the corresponding original stream is cold. Patch streams contain even smaller units (*e.g.,* a single GoP frame), which still fetches the entire stream on midgress, making the MER even higher. These inefficiencies can be also mitigated by stream aggregation, but with more aggressive aggregation (*e.g.,* lower thresholds for aggregation).

2.5.4 *Frozen stream sparsely distribute across partitions.*

In large-scale CDNs serving global users, streams may be delivered by edge nodes across different regions and ISPs. Since cross-ISP and cross-region traffic is restricted [37–40], typical designs treat different ISPs and regions differently (*e.g.,* using different parameters or algorithms), including stream aggregation [16].

Frozen streams distribute sparsely, making them hard to aggregate. We analyze the distribution of streams across *partitions*, where a partition corresponds to a region-ISP pair. For each stream, we measure the number of partitions it spans and the average number of sessions per partition, and then aggregate the results by stream hotness in Fig. 6. We find that frozen streams distribute extremely sparsely: streams with $h(s) < 10$ average fewer than two sessions in each partition they appear in. Frozen streams exhibit high MER and ideally should be aggregated (§2.5.1), but their sparsity renders within-partition aggregation ineffective, while cross-partition aggregation incurs undesirable cross-ISP or cross-region midgress. Thus, frozen streams call for techniques beyond aggregation to reduce MER.

Offloading frozen streams to multihomed nodes. As shown in Tab. 1, multihomed CDN nodes with cross-partition connectivity can offload entire frozen streams, even when they are sparsely distributed across nodes. Although multihomed nodes are more expensive than regular edge nodes, they connect directly to the CDN origin without introducing additional midgress. In contrast, serving frozen streams from regular nodes typically incurs significant midgress traffic (§2.5.1). Thus, offloading frozen streams to multihomed nodes can be cost-effective overall.

Finding 2: Cold streams dominate the population and incur inefficient midgress. Aggregating them onto fewer nodes effectively reduces MER (§2.5.1).

Finding 3: Hot streams are cold at some nodes, and incur high MER, requiring finer-grained aggregation (§2.5.2).

Finding 4: Substreams and patch streams have high MER, requiring more aggressive aggregation (§2.5.3).

Finding 5: Frozen streams, which also incur inefficient midgress, cannot be mitigated by aggregation-based approaches. Multihomed nodes can be helpful (§2.5.4).

3 HCDN Design Guideline

3.1 Augmenting the CDN edge and scheduling streams

The findings in CDN bandwidth analysis (§2.2∼2.5) imply two elements to further reduce relative CDN bandwidth $\overline{P}$:

Augmenting the CDN edge nodes. We can augment the CDN edge with two types of nodes: *(i)* Best-effort edge nodes, which lower the average egress price $\overline{P_E}$ (Finding 1), and *(ii)* Multihomed nodes, which provide cross-partition routing flexibility, thereby enabling the offloading of frozen streams to reduce the inefficiently high MER (Finding 5).

Scheduling streams. We can schedule streams to carefully aggregate them onto a smaller subset of CDN edge nodes to serve (Findings 2∼4), as well as offloading them to the newly

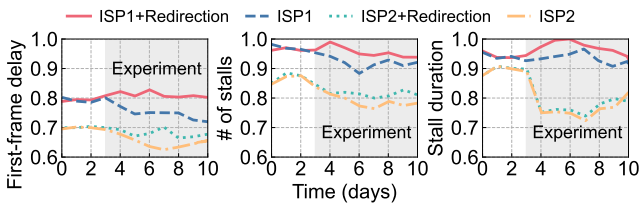


Figure 7: Redirections introduce small but non-trivial QoE degradations. All three metrics use normalized values.

introduced best-effort nodes and multihomed nodes (Findings 1 and 5). In industry practice, CDN stream scheduling is often realized through HTTP redirection for its fast, fine-grained control, in contrast to the coarser and slower adaptation of DNS-based [22] or Anycast [23, 24] methods.

Putting both together, we can conclude the guideline: *Augmenting the CDN edge with best-effort nodes and multihomed nodes, and scheduling streams to map requests to suitable nodes, including those newly added.*

3.2 Challenges towards Practical Deployment

3.2.1 QoE Concerns

Redirections underpin our design but can degrade QoE by introducing an extra client–server roundtrip: the client first receives a redirect, then issues a second request. This added step increases latency in live video streaming, potentially leading to higher playback delay and more stalls. We evaluate this impact using an end-to-end online A/B test that measures first-frame delay, stall count, and stall time. As shown in Fig. 7, the experiment spans two ISPs: `ISP1`) and `ISP2`. For each ISP, we select two edge servers in the same node with comparable traffic and record QoE metrics over ten adjacent days. The A/B test begins on day three (marked "Experiment" in Fig. 7): one server enables redirection (`+Redirection`), while the other disables it. Before the experiment (days $0 \sim 3$), QoE metrics for each server pair remain closely aligned, confirming a controlled environment. During the experiment (days $3 \sim 10$), redirection increases first-frame latency by $2.0 \sim 12.3\%$ on China Mobile and $6.5 \sim 11.4\%$ on China Telecom. It also raises stall counts by $1.8 \sim 7.5\%$ and $0.3 \sim 6.9\%$, and extends stall durations by $1.9 \sim 6.0\%$ and $0.3 \sim 3.8\%$, respectively. In short, redirections introduce modest but non-trivial QoE degradations that must be carefully mitigated.

3.2.2 Long-Term Strategy Management

Over the past four years, our strategy set has continuously evolved. We add new strategies and refine existing ones as we introduce new node types and uncover new inefficiencies. At the production scale, both the development and refinement of strategies are costly, as scheduling affects clients, edge servers, and the central control plane, and every change must pass rigorous testing and review. This is further complicated as multiple strategies may apply to the same stream–node pair, leading to overlapping or conflicting decisions. Additionally, for our production across 500+ clusters, scheduling can incur substantial control-plane overhead in *(i)* edge memory, *(ii)* central memory, and *(iii)* control bandwidth. Therefore, strategy management a persistent challenge in large-scale, in-production CDNs.

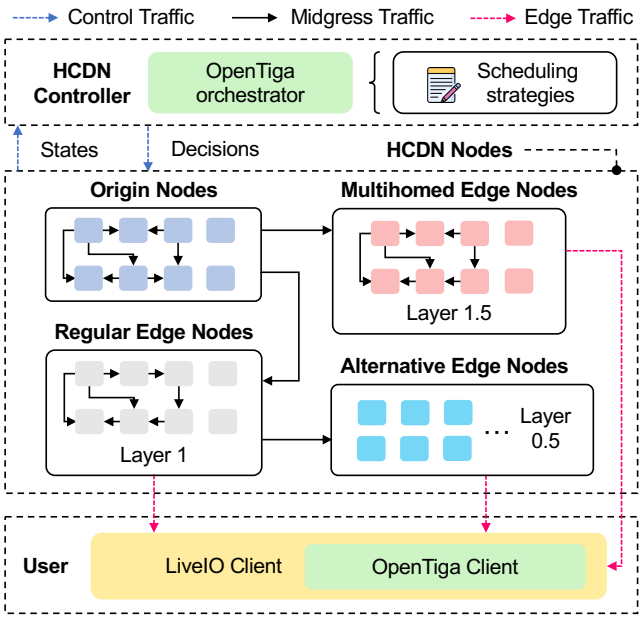


Figure 8: The HCDN architecture.

4 HCDN Design

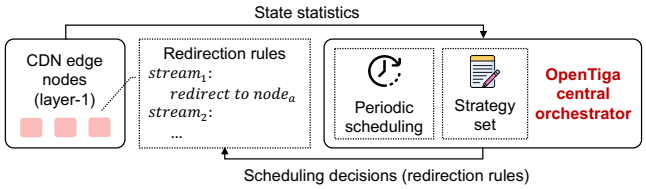
The HCDN system consists of three parts. First, as the guideline suggests, we augment the CDN edge by incorporating new edge resources (§4.1). Second, to address the challenges and enable practical CDN-scale deployment, we design and implement the OpenTiga scheduling framework to coordinate stream scheduling strategies as well as client support (§4.2). Third, building on this framework, we design and implement the stream scheduling strategies (§4.3).

4.1 HCDN Architecture

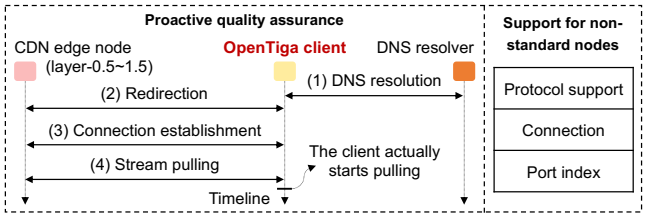
As shown in Fig. 8, the HCDN extends our prior CDN architecture (§2.1). First, it augments the CDN edge by introducing an *alternative edge* that leverages cost-effective best-effort resources, and a *multihomed edge* consisting of nodes connected to multiple ISPs. For clarity, we denote the original CDN edge as *layer 1*, the alternative edge as *layer 0.5*, and the multihomed edge as *layer 1.5*. Second, it augments the control plane by integrating the OpenTiga framework into both the CDN controller and the LiveIO client.

4.1.1 Augmented CDN Edge Nodes

Alternative Edge (Layer 0.5). The alternative edge leverages low-cost resources, such as low-end servers operating on non-standard ports. They vary in type and provide different levels of service quality. For privacy and quality reasons, our CDN does not use any PCDN node. Instead, we use ISP edge nodes. They handle part of the traffic to reduce cost and



(a) OpenTiga central orchestrator



(b) Functionalities of OpenTiga client

Figure 9: OpenTiga scheduling framework

improve efficiency. If a requested stream is unavailable at the alternative edge, it is fetched via midgress from the CDN.

Multihomed Edge (Layer 1.5). Multihomed nodes that connect to multiple ISPs, mainly accounting for frozen streams, which exhibit high MER (§2.5.1) but are too sparsely distributed to benefit from aggregation (§2.5.4). Multihomed nodes bypass these restrictions through their cross-ISP and cross-region capability [17–19]. By connecting directly to the CDN origin, this approach mitigates the high MER of frozen streams despite their sparsity.

4.1.2 *Augmented Controller and Client.*

The CDN controller and LiveIO client are augmented to accommodate the OpenTiga scheduling framework (§4.2) and the scheduling strategies (§4.3).

Controller. The controller is extended to incorporate the OpenTiga orchestrator (§4.2.1) and enforce all scheduling strategies (§4.3). It collects system statistics from the CDN (*e.g.,* hotness $h(s,n)$, load of CDN nodes), and feeds them to the OpenTiga orchestrator. The orchestrator periodically recalculates scheduling decisions, which the controller then distributes to the layer-1 CDN edge nodes.

Client. The client is extended to incorporate the OpenTiga client (§4.2.2), which implements proactive QoE assurance techniques and supports non-standard nodes in layer 0.5.

4.2 OpenTiga Scheduling Framework

As shown in Fig. 9, the OpenTiga scheduling framework consists of two components: *(i)* OpenTiga *central orchestrator* (§4.2.1) and *(ii)* OpenTiga *client* (§4.2.2), integrated into the HCDN controller and the LiveIO client, respectively. The orchestrator coordinates multiple scheduling strategies and periodically generates scheduling decisions, while the client applies quality assurance techniques and ensures compatibility with heterogeneous edge resources.

4.2.1 OpenTiga *Central Orchestrator*

The orchestrator, as shown in Fig. 9a, resides in the controller of HCDN. It periodically collects required state statistics from CDN edge nodes (*e.g.,* as stream hotness, node hotness, node capacity, and load) and feeds them into the corresponding strategies. Once the scheduling decisions (*i.e.,* redirection rules) are derived, the orchestrator distributes them to the relevant CDN edge nodes.

Unified Abstraction. The orchestrator defines a unified abstraction, which is shared by all scheduling strategies:

```
// Get states from a subset of streams.
Func get_stream_states(void);
// Split a stream's nodes into partitions.
Func get_partitions(void);
// Select a subset of streams to schedule.
Func select_streams(partition);
// Sources and targets for redirection.
Func get_src_and_tgt(stream, part);
```

Each strategy exposes four primitives, corresponding to the main steps of scheduling: *(i) State collection*, which retrieves states from a subset of streams. The overhead is reduced by restricting state collection to only a subset of streams. *(ii) Node partitioning*, which divides CDN edge (layer-1) nodes into partitions (*e.g.,* ISP_1-Region_1). *(iii) Stream selection*, which determines the streams to schedule within each partition; *(iv) Source and target derivation*, which specifies the source and target nodes for redirection.

Periodic Scheduling. The orchestrator periodically executes all scheduling strategies in sequence at a 15-second interval. Each scheduling iteration proceeds as follows:

```
1  ON timer_tick():
2    FOR s in strategies:
3      stats <- s.get_stream_states();
4      FOR p in s.get_partitions():
5        FOR st in s.select_streams(p):
6          decisions = {};
7          src, tgt <- s.get_src_and_tgt(st, p);
8          for sr in src:
9            decisions[st,sr] <- select(tgt)
10       FOR (st,sr), tg in decisions:
11         send_decision(sr, rule(st, tg));
```

For each strategy *s*, the orchestrator invokes all its methods as defined by the unified abstraction. It first collects states of streams specified by the strategy (line 3). Then, it partitions all layer-1 CDN edge nodes into groups according to the strategy (line 4). Partitioning is sometimes trivial, since certain strategies operate globally (*e.g.,* frozen stream aggregation, as described in §4.3.1). Then, for each stream *st* in each partition *p*, the strategy derive source nodes and target nodes for redirection (line 7). The orchestrator then generates a set of redirection rules, each mapping a stream from a layer-1 CDN edge node to a target node in layer-0.5∼1.5

(lines 8∼9). Finally, it distributes the resulting scheduling decisions to the corresponding source nodes (lines 10∼11).

Managing Strategies. The unified abstraction handles the possible conflicts of strategies. Under the unified abstraction, each strategy implements the standard set of methods. The orchestrator inspects each strategy's implemented methods, focusing on their parameters and outputs, to check whether the resulting selections and and (stream, source node) pairs intersect across strategies. If any intersection is detected, the orchestrator suppresses overlapping redirection rules, ensuring that the final decision set remains disjoint.

Modular Strategy Development. The unified abstraction enables modular strategy development by decoupling evolving scheduling policy from the stable core mechanism. In our earlier practice, introducing or updating a strategy required coordinated changes across client logic, edge servers, and the control plane (§3.2.2). By contrast, strategies are now implemented as lightweight Lua [41] scripts that invoke the exposed APIs. This design transforms strategy development from invasive codebase modifications into safe scripting tasks, allowing operators to sustain operational agility and iterate on logic without destabilizing the underlying infrastructure. Consequently, we can readily instantiate new stream scheduling strategies and update them based on observed production performance.

Reducing Control Data Overhead. The fine-grained state statistics are required for fine-grained stream aggregation. This is suggested by the MER analysis of hot streams, which are cold on certain nodes yet hot on others, leading to inefficiently high MER and requiring aggregation from cold nodes onto hot nodes (§2.5.2). However, cold streams, which are the majority, do not require fine-grained statistics. Hot streams, although spanning many nodes and needing detailed records, account for only a small fraction of streams overall. Therefore, we coordinate hot-stream and cold-stream aggregation strategies by carefully configuring thresholds so that cold streams rely only on coarse-grained state, while hot streams require fine-grained state but remain small in count, keeping the total overhead low.

4.2.2 OpenTiga *Client*

Proactive Quality Assurance. In the default workflow of live video streaming, the client must sequentially perform the following steps on the critical path: *(i) DNS resolution.* The client resolves the stream's domain name into the IP address of a CDN node. *(ii) Redirection.* The client sends a request to this node, which either returns a server address directly or issues a redirect to another CDN node. *(iii) Connection establishment.* The client then establishes a transport connection with the assigned server's address. *(iv) Stream pulling.* Finally, the client requests the video stream, triggering content fetching and delivery. Since all four stages occur only after the user initiates playback, they directly add to startup latency and increase the probability of stalls. To mitigate this issue, we adopt a client-CDN co-design approach that aligns with common industry practices [27,28]. Specifically, the OpenTiga client leverages the recommendation feed commonly integrated into live streaming applications, which presents streams in a deterministic order. The client follows this feed to identify the upcoming stream and proactively executes the above stages in advance. As Fig. 9b shows, DNS resolution, redirection, and connection establishment are shifted off the critical path, and a pre-pull request ensures that video content is cached or warmed up at the edge. Consequently, when playback starts, the stream begins with minimal delay, and proactive quality assurance significantly reduces startup latency while lowering the likelihood of stalls, thereby improving overall QoE. Furthermore, to ensure robustness against failures of target nodes, the client immediately aborts the attempt and falls back to the default DNS-resolved node if the target node is unresponsive.

Compatibility for Non-Standard Edge Resources. Although cost-effective, some best-effort edge nodes operate on non-standard ports rather than the default HTTP/HTTPS ports. Utilizing these ports faces practical ecosystem constraints, as strict ISP firewalls or middleboxes often block non-standard traffic or flag non-standard HTTPS as untrusted. To ensure seamless connectivity, the OpenTiga client extends its connection management to support arbitrary port numbers. Specifically, the client *(i)* parses and stores port information contained in the CDN's redirection response, *(ii)* establishes transport connections over the indicated non-standard ports, and *(iii)* ensures security policy compliance through protocol and certificate validation. Unlike standard HTTP requests that rely on default ports without port authentication, our method explicitly integrates port specification and certificate verification to enhance reliability and security. This approach enables safe and compatible utilization of heterogeneous edge resources.

4.3 Scheduling Strategies

As shown in Fig. 10, we deploy four strategies: *(i) frozen stream offloading* (§4.3.1), *(ii) cold stream aggregation* (§4.3.2), *(iii) hot stream aggregation* (§4.3.3), and *(iv) hot stream offloading* (§4.3.4). All strategies operate on streams served by layer-1 CDN edge nodes. Strategies *(ii)* and *(iii)* aggregate streams within layer-1, whereas *(i)* and *(iv)* offload them to layer-1.5 and layer-0.5 edge nodes. We now describe each strategy in detail, including the design considerations and scheduling steps under the unified abstraction.

4.3.1 *Frozen Stream Offloading*

We offload frozen streams to layer-1.5 multihomed edge nodes (strategy 1 in Fig. 10) to reduce inefficient midgress leveraging the cross-partition capability of layer-1.5 nodes.

Scheduling Steps.

• *State Collection.* The strategy relies on stream hotness $h(s)$ for all streams and CPU/memory load of layer-1.5 nodes.

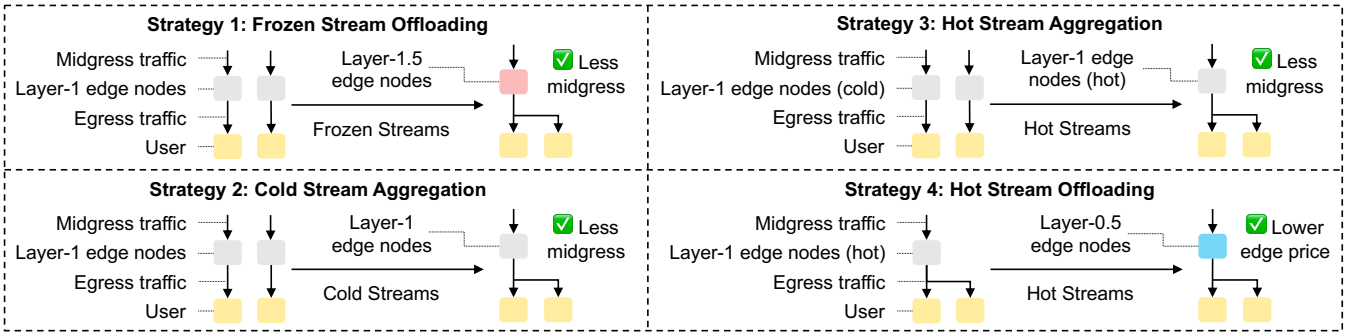


Figure 10: Stream scheduling strategies in HCDN.

• *Node Partitioning.* Because layer-1.5 CDN edge nodes are multihomed, the strategy treats streams across ISPs and regions as a single group. Node partitioning is therefore trivial, with one partition containing all layer-1 CDN edge nodes.

• *Stream Selection.* It selects frozen streams by a hotness threshold k_1, selecting all streams s such that $h(s) < k_1$.

• *Source and Target Derivation.* Frozen streams migrate from l_1 single-homed layer-1 edge nodes to m_1 multihomed layer-1.5 nodes, where l_1 and m_1 are pre-defined parameters. Targets are further selected as nodes whose CPU and memory loads remain below thresholds cpu_1 and mem_1.

4.3.2 Cold Stream Aggregation

The cold stream aggregation strategy redirects each selected cold stream from the layer-1 CDN edge nodes that serve it onto a small subset of these nodes (strategy 2 in Fig. 10).

Coarse-grained Aggregation. Following the coordination method of OpenTiga (§4.2.1), this strategy aggregates streams using only their hotness $h(s)$. Source and target nodes are selected based solely on their CPU and memory load. This simplification is justified because, for cold streams, most nodes exhibit low $h(s,n)$, and random selection of sources and targets can still achieve substantial gains.

Handling Heterogeneous Stream Types. Substreams require more aggressive aggregation (§2.5.3). To accommodate this heterogeneity, we apply distinct threshold configurations for different stream types. Specifically, we assign lower hotness thresholds for substreams, marking them as cold earlier and enabling more aggressive aggregation.

Scheduling Steps.

• *State Collection.* The cold stream aggregation strategy collects stream hotness $h(s)$ for streams. It also collects CPU/memory load of all layer-1 CDN edge nodes.

• *Node Partitioning.* The strategy partitions layer-1 CDN edge nodes by ISP and province.

• *Stream Selection.* The strategy applies to cold streams (excluding frozen ones), identified using a per-partition hotness threshold $k_2(p)$ and the global frozen-stream threshold k_1. It selects all streams s such that $k_1 < h(s) < k_2(p)$.

• *Source and Target Derivation.* For a cold stream s within partition p, we randomly select $l_2(p)$ nodes from the set of nodes hosting s in p as redirection sources, and then select $m_2(p)$ nodes from the remaining nodes as redirection targets.

4.3.3 Hot Stream Aggregation

We aggregate and migrate hot streams from "cold nodes" to "hot nodes" that already serve them within layer-1 CDN edge nodes (strategy 3 in Fig. 10).

Finer-Grained Aggregation. This strategy addresses inefficient midgress caused by hot streams served on their "cold nodes" with low stream-node hotness $h(s,n)$ (§2.5.2). Unlike cold stream aggregation, it does not treat all nodes serving a stream as identical. Instead, it leverages stream-node hotness $h(s,n)$ to distinguish between "cold" and "hot" nodes, and migrates streams from the former to the latter. By consolidating requests onto hot nodes, future requests are served by fewer nodes, allowing cold nodes to drain and become idle, thereby reducing midgress overhead. Similar to cold stream aggregation, this strategy also accounts for heterogeneous stream types and partition-level midgress imbalance.

Scheduling Steps.

• *State Collection.* Following the OpenTiga framework's method of reducing control data overhead (§4.2.1), the strategy first collects coarse-grained stream hotness $h(s)$ for all streams, and then fine-grained stream-node hotness $h(s,n)$ only for hot streams with $h(s) > k_2(p)$. This threshold is identical to that for cold stream aggregation, to achieve seamless inter-strategy coordination.

• *Node Partitioning.* Hot stream aggregation also partitions all layer-1 CDN edge nodes into ISP-province pairs.

• *Stream Selection.* This strategy targets hot streams, complementing cold stream aggregation so that together the two strategies cover the full hotness range. Thresholds for heterogeneous stream types are aligned with those in cold stream aggregation to ensure seamless coverage.

• *Source and Target Derivation.* For each stream s, the strategy selects the coldest $l_3(p, s.type)\%$ of nodes hosting s as source nodes, and the hottest $m_3(p, s.type)\%$ as target nodes.

Thresholds are further adapted to stream type: substreams are assigned higher $l_3(\cdot)$ and lower $m_3(\cdot)$ values, enabling more aggressive aggregation as suggested by §2.5.3.

4.3.4 Hot Stream Offloading

To fully explore the opportunity to reduce high bandwidth cost at layer-1 CDN edge nodes (§2.3), we offload streams to layer-0.5 alternative edge nodes (strategy 4 in Fig. 10). These alternative nodes rely on cost-effective best-effort nodes to lower the average unit price of egress bandwidth.

Only offloading hot streams to reduce additional midgress traffic. When a user request is redirected to a layer-0.5 node where the stream is not yet available, the node must first fetch the stream from a layer-1 edge node via midgress. Since cold streams generally exhibit higher MER than hot streams (§2.5.1), offloading cold streams (or cold stream-node pairs with low $h(s,n)$) incurs high additional midgress overhead. To mitigate this, we offload only hot streams from hot nodes (*i.e.,* stream-node pairs with high $h(s,n)$), subject to layer-0.5 capacity constraints. This keeps additional midgress traffic low while reducing egress cost.

Scheduling Steps.

• *State Collection.* The strategy collects stream hotness $h(s)$ for all streams and stream-node hotness $h(s,n)$ for only hot streams with hotness $h(s) > k_4(p)$.

• *Node Partitioning.* We partition layer-1 CDN edge nodes by ISP-province pairs in practice.

• *Stream Selection.* The strategy selects hot streams s with $h(s) > k'_4(p)$, where $k'_4(p)$ is a predefined hotness threshold.

• *Source and Target Derivation.* For each selected stream s in partition p, we first identify all layer-1 CDN edge nodes hosting s and randomly select $l_4(p)$ of them as redirection sources. We then choose $m_4(p)$ layer-0.5 alternative edge nodes in the same partition as redirection targets.

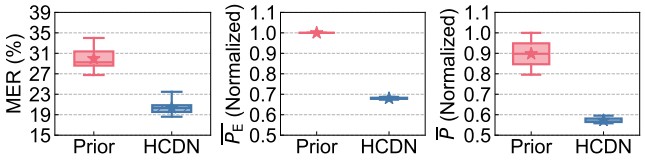
4.3.5 Parameters for Stream Scheduling Strategies

The strategies involve a set of pre-defined parameters: k_1, l_1, m_1, $\{k_i(\cdot)\}_{i=2,4}$, $\{l_i(\cdot)\}_{i=2,3,4}$, $\{m_i(\cdot)\}_{i=2,3,4}$, $\{cpu_i\}_{i=1,2,3,4}$, $\{mem_i\}_{i=1,2,3,4}$, and $k'_4(\cdot)$. With hundreds of partitions (*i.e.,* region-ISP pairs) globally, the number of parameters can reach into the thousands. We configure these parameters using data-driven techniques based on distributions of session counts, bandwidth consumption, and CPU/memory utilization from recent operational logs, then apply regression-based modeling to update the parameters. The update process runs periodically to adapt to shifting workload patterns.

5 Evaluation

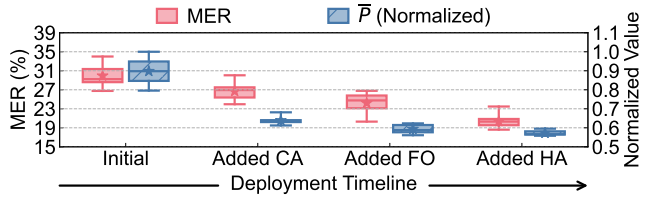
5.1 Implementation and Deployment

During the past four years, we have incrementally implemented and deployed our system design, evolving our prior CDN architecture into HCDN. We augment the CDN edge with cost-effective best-effort nodes, implement the OpenTiga central orchestrator in the CDN controller in Go, and integrate the OpenTiga client into the LiveIO client. The scheduling strategies are realized as Lua [41] scripts built on the APIs of the OpenTiga scheduling framework, which simplifies updates and the addition of new strategies. HCDN spans across > 500 nodes and supports major live streaming events. This long-term, large-scale deployment allows us to evaluate its effectiveness in reducing CDN bandwidth cost while remaining lightweight and maintaining QoE (§5.2∼5.4), and to share our lessons learned (§6).

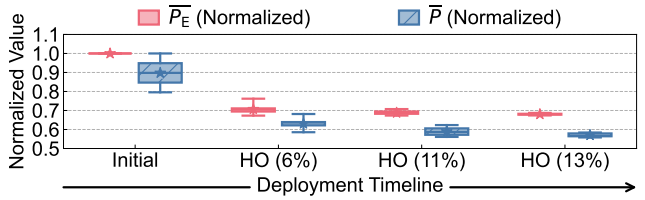


(a) Midgress-egress ratio (MER). (b) Average edge price ($\overline{P_E}$). (c) Relative bandwidth cost ($\overline{P}$).

Figure 11: HCDN effectively reduces the MER, $\overline{P_E}$, and $\overline{P}$. ⋆ indicates mean value.



(a) Accumulative reduction of MER.



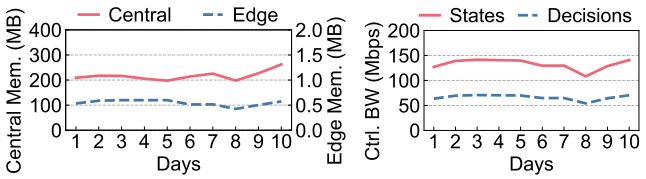
(b) Accumulative reduction of $\overline{P_E}$.

Figure 12: Cumulative cost-reduction gains from incremental deployment of four strategies. "⋆" indicates mean value.

Evaluation Methodology. We evaluate HCDN in our current large-scale production environment, comparing it to the prior architecture using online operation logs. Since we have already migrated to HCDN, we use newly collected logs from HCDN and historical logs from the prior architecture. Our study focuses on four aspects: *(i)* the reduction of CDN bandwidth cost compared to the prior architecture (§5.2), detailing the impact of individual strategies along the incremental deployment timeline, *(ii)* the system overhead incurred (§5.3), *(iii)* the preservation of user QoE (§5.4).

5.2 Bandwidth Cost Reduction

We quantify the reduction in bandwidth cost (Fig. 11). HCDN lowers the average MER from 0.30 to 0.20, representing a 33% reduction compared to our prior CDN archi-



(a) Memory overhead. (b) Control bandwidth.

Figure 13: System overhead.

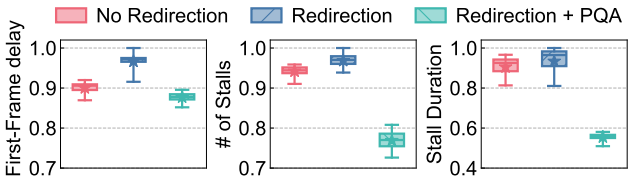


Figure 14: QoE metrics. "⋆" indicates mean value.

tecture (Fig. 11a). The average edge unit price $\overline{P_E}$ decreases by 32% (Fig. 11b). Overall, the relative bandwidth cost $\overline{P}$ drops by 36% compared to the prior architecture (Fig. 11c). These results demonstrate that HCDN's augmented CDN edge and stream scheduling strategies effectively reduce the bandwidth cost. Notably, $\overline{P}$ decreases more than MER and $\overline{P_E}$, because the average midgress price $\overline{P_{Mid}}$ also declines as more nodes are migrated to the flat topology (§2.4).

Incremental Deployment. As shown in Fig. 12, we summarize the incremental deployment and cumulative benefits of the four stream scheduling strategies: cold stream aggregation (CA), frozen stream offloading (FO), hot stream aggregation (HA), and hot stream offloading (HO). The benefits of introducing layer-0.5 alternative edge nodes and layer-1.5 multihomed edge nodes are captured by FO and HO, since these strategies explicitly schedule streams to these nodes.

• *Benefits in MER.* The CA, FO, and HA strategies, which aim to reduce MER, are deployed sequentially over the past four years (Fig. 12a). Initially, when none of these strategies is deployed, the MER averages 0.30. After sequentially deploying CA, FO, and HA, the MER decreases step by step to 0.27, 0.24, and 0.20, with each new strategy contributing noticeable improvements. Corresponding to the three deployment steps, the relative bandwidth cost $\overline{P}$ also decreases to 71%, 66%, and 64% of the initial value.

• *Benefits in* $\overline{P_E}$. We incrementally deploy HO to reduce the average egress price $\overline{P_E}$ by gradually adding layer-0.5 alternative CDN edge nodes (Fig. 12b). As we expand the deployment of HO (*i.e.,* progressively scheduling 3%, 11%, and 13% of the total traffic to layer-0.5), $\overline{P_E}$ decreases step by step to 71%, 69%, and 68% of the initial value, while the relative bandwidth cost $\overline{P}$ also drops correspondingly.

5.3 System Overhead

In Fig. 13, we demonstrate that the overhead incurred by our system design is acceptable. This analysis is based on production logs collected over 10 consecutive days.

Memory Overhead. The OpenTiga central controller primarily stores aggregated state statistics (*e.g.,* hotness), along with the scheduling strategies currently in effect. Since the controller keeps fine-grained stream-node hotness only for hot streams and coarse-grained hotness for others, the central controller memory footprint remains moderate at 197 ∼ 262 MB (Fig. 13a). Each edge node keeps only its local statistics and the subset of strategies relevant to its operation, incurring only 0.42 ∼ 0.60 MB per edge node (Fig. 13a).

Control Bandwidth Overhead. The control traffic involves periodically transmitting state statistics from edge nodes to the controller and disseminating scheduling decisions back. As shown in (Fig. 13b), the traffic volumes are below 150 and 80 Mbps for states and decisions, respectively. This is negligible compared to typical inter-datacenter capacity.

5.4 QoE Performance

We demonstrate that HCDN maintains user QoE despite its redirection-based stream scheduling strategies. As shown in Fig. 14, we evaluate three QoE metrics: first-frame delay, stalls per session, and stall duration per session, under three settings: *(i)* no redirection, *(ii)* redirection, and *(iii)* redirection with proactive quality assurance (PQA) in the OpenTiga client. Redirection without PQA increases the three metrics by 7.4%, 2.8%, and 3.2%, respectively, compared to no redirection. By contrast, adding PQA reduces the metrics by 9.2%, 21%, and 41% relative to redirection without PQA. Even compared to the baseline without redirection, redirection with PQA still achieves improvements of 2.6%, 18%, and 39%. These gains arise because PQA proactively initiates the whole stream pulling process, which not only masks the redirection delay by performing redirection in advance, but also reduces overall latency by initiating connection establishment and stream fetching beforehand (§4.2.2).

6 Lessons Learned

Fluctuating stream hotness distribution. In practice, the distribution of stream hotness fluctuates significantly, not only because individual streams vary in hotness but also because customers dynamically decide which streams to assign to our CDN versus others. When mostly cold streams are allocated, we observe a higher proportion of cold traffic, and conversely, hot streams dominate when they are primarily assigned. An excessive share of cold streams raises relative cost due to high MER. To mitigate this, our data operations platform continuously monitors the hot–cold ratio and adaptively adjusts scheduling parameters in a data-driven manner to prevent performance degradation.

Scalability beyond CDN Capacity. Our evaluation shows that the system overhead of HCDN is acceptable for production-scale CDNs, allowing it to scale effectively to large deployments. When demand exceeds available bandwidth, we selectively offload traffic to third-party CDNs

through federated CDN infrastructures [42, 43]. We leverage a capacity model built on physical resource constraints, which continuously monitors resource utilization, and offload traffic when the capacity risks are exceeded.

Falling Back to DNS Scheduling. During large-scale events (*e.g.,* the Asian Games or the World Cup), customers may request that critical streams bypass redirection and instead fall back to the default nodes determined by DNS to ensure stability. However, popular streams often experience sudden surges in viewership, causing extremely high instantaneous concurrency at specific nodes. In practice, DNS caching delays traffic migration, leaving overloaded nodes unable to absorb the surge and severely degrading service quality (*e.g.,* reduced pull success rate). To address this, we integrate real-time load detection and rapid traffic migration. When overload is detected, redirection-based scheduling immediately diverts new traffic to non-overloaded nodes.

Adapting to Cross-Province Restrictions. In practice, some ISPs charge for traffic exported across provincial boundaries and cap cross-province traffic ratios (*e.g.,* 70%). To control cost, HCDN falls back to DNS-assigned default nodes in affected provinces when caps are tight. Some ISPs also impose QoE-based restrictions. In this case, we monitor cross-province ratios and node connectivity (*e.g.,* success rates) in real time and dynamically reduce cross-province redirections or disable low-quality nodes.

Midgress Persistence. Unlike traditional CDNs for video-on-demand or web content, live video streaming CDNs configure a midgress persistence time specifying how long the midgress traffic stays active after streams become idle. Longer persistence avoids reconnection, letting the edge respond to returning viewers immediately and improving startup quality, but also wastes bandwidth when no users are present. Persistence time is thus a key knob – longer favors QoE, shorter favors cost. In practice, we tune it conservatively, reducing persistence after inactivity to save bandwidth while maintaining acceptable user experience.

7 Discussion

Avoiding Negative Effects on QoE. In HCDN, the negative impact of redirection on QoE is mitigated by proactive quality assurance in the OpenTiga client (§4.2.2), which leverages the recommendation system of the LiveIO client. However, in more general real-time streaming scenarios (*e.g.,* cloud gaming [44–47], VR [48, 49], video conferencing [50, 51]) where the client lacks the ability to predict the next live stream, one must unfortunately trade off degraded QoE against the benefits gained from redirection. In such cases, it may be useful to model and predict the negative impact of each redirection and selectively perform redirection.

Generalizing to Other Objectives. While we focus on bandwidth cost in this paper, the HCDN architecture readily supports other scheduling goals. Since its strategies are defined through a unified abstraction in the OpenTiga framework, new objectives can be incorporated simply by instantiating the same set of primitives with metrics tailored to the desired goal. For example, HCDN can optimize latency by integrating real-time measurements and enforcing strict SLOs [52–55], or target other widely studied objectives such as energy efficiency [56] and load balance [57].

8 Related Work

CDN Architecture Design. Early CDNs rely on hierarchical overlays [58, 59], which improve QoE by reducing path stretch and ensuring stable delivery [60–62]. However, hierarchy creates scalability and hot-spot issues, and flat relay architectures [9, 32, 33] address this issue by organizing nodes into a flat circular overlay where each acts as edge, origin, and relay. HCDN is built on a hybrid of hierarchical and flat designs. Some works introduce cost-effective edge nodes [11, 13–15, 20, 58, 59, 63] to reduce cost, which we also integrate into HCDN as layer-0.5 alternative edge nodes. HCDN's alternative edge nodes serve as a specific type of cost-effective edge nodes.

CDN Scheduling. DNS-based scheduling [22] maps users to edge nodes by network location, while anycast-based scheduling [23, 24] relies on BGP to steer traffic. Both are network-centric and coarse-grained, ignoring fine-grained application goals. Redirection-based scheduling [16] is closer to our work. We extend it by: *(i)* augmenting its strategies by new strategies redirecting requests to new types of nodes and mitigating newly discovered inefficiencies, and *(ii)* preventing negative effects of redirection on QoE. Other scheduling paradigms are complementary to our design. Overlay scheduling [9, 10, 64] targets core paths instead of the edge. C3 [65] assigns clients across multiple CDNs.

9 Conclusion

We present HCDN, a hybrid CDN architecture that masters the growing complexity and massive bandwidth costs of large-scale live video streaming. By abstracting heterogeneous edge resources and fine-grained stream dynamics into a unified scheduling framework, HCDN replaces ad-hoc stream scheduling with principled, coordinated orchestration. Our multi-year production deployment demonstrates that HCDN achieves substantial cost savings while maintaining acceptable control overhead and video quality. HCDN provides a practical case for robust, modular and scalable stream scheduling in live video delivery.

Acknowledgments

We thank our shepherd, Lakshminarayanan Subramanian, and the anonymous NSDI reviewers for their insightful comments and suggestions. This work is supported by National Key Research and Development Plan, China (Grant No. 2023YFB2903902) and the National Natural Science Foundation of China (Grant No. U25B2040, 62272010). Jing Liu and Chenren Xu are the corresponding authors.

References

[1] Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. https://futuretimeline.net/data-trends/pdfs/cisco-2017-2022.pdf.

[2] demandsage. 42 live streaming statistics 2025: Trends growth. https://www.demandsage.com/live-streaming-statistics.

[3] Douyin live homepage. https://www.douyin.com/, 2025.

[4] Taobao live homepage. https://taolive.taobao.com, 2025.

[5] Youtube live homepage. https://www.youtube.com/live, 2025.

[6] Facebook live homepage. https://www.facebook.com/watch/live/, 2025.

[7] Twitch live homepage. https://www.twitch.tv/, 2025.

[8] Kuaishou live homepage. https://www.kuaishou.com/, 2025.

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

[10] Matthew K Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. In *ACM SIGCOMM*, 2015.

[11] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, and Heung-Keung Chai. Analysis of a cdn–p2p hybrid architecture for cost-effective streaming media distribution. *Multimedia Systems*, 11(4), 2006.

[12] Kuaishou Technology. Announcement of the results for the three months ended march 31, 2025. https://www1.hkexnews.hk/listedco/listconews/sehk/2025/0527/2025052700279.pdf.

[13] Rui-Xiao Zhang, Haiping Wang, Shu Shi, Xiaofei Pang, Yajie Peng, Zhichen Xue, and Jiangchuan Liu. Enhancing resource management of the world's largest PCDN system for On-Demand video streaming. In *USENIX ATC*, 2024.

[14] Chenfei Tian, Shaorui Ren, Yixuan Zhang, and Mingwei Xu. Semi-coupled congestion control for multi-site parallel downloading. In *ACM MMSys*, 2023.

[15] Ge Zhang, Wei Liu, Xiaojun Hei, and Wenqing Cheng. Unreeling xunlei kankan: Understanding hybrid cdn-p2p video-on-demand streaming. *IEEE Transactions on Multimedia*, 2015.

[16] Rui-Xiao Zhang, Changpeng Yang, Xiaochan Wang, Tianchi Huang, Chenglei Wu, Jiangchuan Liu, and Lifeng Sun. Aggcast: Practical cost-effective scheduling for large-scale cloud-edge crowdsourced live streaming. In *ACM MM*, 2022.

[17] David K Goldenberg, Lili Qiu, Haiyong Xie, Yang Richard Yang, and Yin Zhang. Optimizing cost and performance for multihoming. *ACM SIGCOMM CCR*, 34(4), 2004.

[18] DataPacket. Multihomed network vs single-homed network. https://www.datapacket.com/blog/multihomed-network-vs-single-homed-network, 2024.

[19] Aditya Akella, Bruce Maggs, Srinivasan Seshan, Anees Shaikh, and Ramesh Sitaraman. A measurement-based analysis of multihoming. In *ACM SIGCOMM*, 2003.

[20] Huanhuan Zhang, Congkai An, Anfu Zhou, Yifan Zhu, Weilin Sun, Yixuan Lu, Jiahao Chen, Liang Liu, Huadong Ma, and Aiguo Fei. Venus: Enhancing qoe of crowdsourced live video streaming by exploiting multiflow viewer assistance. In *ACM MobiCom*, 2024.

[21] MMSys'23 Grand Challenge. Multi-site parallel downloading challenge. https://mmsysgc23.github.io, 2023.

[22] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM CCR*, 2015.

[23] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M Maggs, Haiying Shen, Ramesh K Sitaraman, and Xiaowei Yang. Anyopt: Predicting and optimizing ip anycast performance. In *ACM SIGCOMM*, 2021.

[24] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. {FastRoute}: A scalable {Load-Aware} anycast routing architecture for modern {CDNs}. In *USENIX NSDI*, 2015.

[25] Roy T. Fielding and Julian F. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, June 2014.

[26] Brad Cain, Oliver Spatscheck, Abbie Barbir, and Raj Nair. Known content network (cn) request-routing mechanisms. https://www.rfc-editor.org/rfc/rfc3568, 2003. RFC 3568.

[27] lazygeek78. Design of TikTok. https://medium.com/@lazygeek78/design-of-tiktok-e2da18e17f2b, 6 2023.

[28] Chaitanya Ekanayake and Vijay Gondi. Using machine learning to improve streaming quality at Netflix. https://netflixtechblog.com/using-machine-learning-to-improve-streaming-quality-at-netflix-9651263ef09f, 3 2017.

[29] Kartik Hosanagar. Cdn pricing. In *Content delivery networks*, pages 211–224. Springer, 2008.

[30] Kartik Hosanagar, John Chuang, Ramayya Krishnan, and Michael D Smith. Service adoption and pricing of content delivery network (cdn) services. *Management Science*, 54(9):1579–1593, 2008.

[31] Bo Yan, Shu Shi, Yong Liu, Weizhe Yuan, Haoqin He, Rittwik Jana, Yang Xu, and H Jonathan Chao. Livejack: Integrating cdns and edge clouds for live content broadcasting. In *ACM MM*, 2017.

[32] Leonidas Kontothanassis, Ramesh Sitaraman, Joel Wein, Duke Hong, Robert Kleinberg, Brian Mancuso, David Shaw, and Daniel Stodolsky. A transport layer for live streaming in a content delivery network. *Proceedings of the IEEE*, 2004.

[33] Ramesh K Sitaraman, Mangesh Kasbekar, Woody Lichtenstein, and Manish Jain. Overlay networks: An akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services*, 2014.

[34] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. {AdaptSize}: Orchestrating the hot object memory cache in a content delivery network. In *USENIX NSDI 17*, 2017.

[35] Bruce M Maggs and Ramesh K Sitaraman. Algorithmic nuggets in content delivery. *ACM SIGCOMM Computer Communication Review*, 45(3):52–66.

[36] John McCabe. On serial files with relocatable records. *Operations Research*, 13:609–618, 1965.

[37] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *ACM IMC*, 2005.

[38] Zhejiayu Ma, Soufiane Roubia, Frédéric Giroire, and Guillaume Urvoy-Keller. When locality is not enough: Boosting peer selection of hybrid cdn-p2p live streaming systems using machine learning. In *IFIP TMA 2021-Network Traffic Measurement and Analysis Conference*, 2021.

[39] Nazanin Magharei, Reza Rejaie, Ivica Rimac, Volker Hilt, and Markus Hofmann. Isp-friendly live p2p streaming. *IEEE/ACM Transactions on Networking*, 2013.

[40] Ye Tian, Ratan Dey, Yong Liu, and Keith W. Ross. China's internet: Topology mapping and geolocating. In *IEEE INFOCOM*, 2012.

[41] Lua programming language. https://www.lua.org/.

[42] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. C3:{internet-scale} control plane for video quality optimization. In *USENIX NSDI*, 2015.

[43] Alexandros Biliris, Chuck Cranor, Fred Douglis, Michael Rabinovich, Sandeep Sibal, Oliver Spatscheck, and Walter Sturm. Cdn brokering. *Computer Communications*, 2002.

[44] Cloud gaming market: Global industry trends, share, size, growth, opportunity and forecast 2023-2028. IMARC Group Report No. 5732901, 2023.

[45] Google cloud gaming solutions. https://cloud.google.com/solutions/games, 2023.

[46] Samsung gaming hub. https://www.samsung.com/us/televisions-home-theater/tvs/gaming-hub, 2025.

[47] Start cloud gaming. https://start.qq.com/, 2023.

[48] Google maps live view support. https://support.google.com/maps/answer/9332056?hl=en&co=GENIE.Platform%3DiOS, 2023.

[49] Youtube vr home. https://vr.youtube.com/, 2023.

[50] Zoom: One platform to connect. https://zoom.us, 2023.

[51] Microsoft teams group-chat software. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software, 2023.

[52] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving Consistent Low Latency for Wireless Real-Time Communications with the Shortest Control Loop. In *ACM SIGCOMM*, 2022.

[53] Jiangkai Wu, Yu Guan, Qi Mao, Yong Cui, Zongming Guo, and Xinggong Zhang. ZGaming: Zero-Latency 3D Cloud Gaming by Image Prediction. In *ACM SIGCOMM*, 2023.

[54] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *USENIX NSDI*, 2018.

[55] Yuhan Zhou, Tingfeng Wang, Liying Wang, Nian Wen, Rui Han, Jing Wang, Chenglei Wu, Jiafeng Chen, Longwei Jiang, Shibo Wang, Honghao Liu, and Chenren Xu. AUGUR: Practical Mobile Multipath Transport Service for Low Tail Latency in Real-Time Streaming. In *USENIX NSDI*, 2024.

[56] Daibo Liu, Chao Qian, Huigui Rong, Siwang Zhou, Chaocan Xiang, and Hongbo Jiang. Energy and qoe optimization for mobile video streaming with adaptive brightness scaling. *ACM Trans. Sen. Netw.*, 2024.

[57] Youssouph Faye, Francescomaria Faticanti, Shubham Jain, and Francesco Bronzino. Videojam: Self-balancing architecture for live video analytics. In *IEEE/ACM SEC*, 2024.

[58] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky. In *ACM MM*, 2009.

[59] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Livesky: Enhancing cdn with p2p. *ACM TOMM*, 2010.

[60] Vijay K. Adhikari, Yang Guo, Fang Hao, Volker Hilt, Zhi-Li Zhang, Matteo Varvello, and Moritz Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *IEEE/ACM TON*, 2015.

[61] F. Larumbe and A. Mathur. Under the hood: Broadcasting live video to millions. https://engineering.fb.com/2015/12/03/ios/under-the-hood-broadcasting-live-video-to-millions, 2015.

[62] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y. Zhao. Anatomy of a personalized livestreaming system. In *ACM IMC*, 2016.

[63] Huanhuan Zhang, Congkai An, Anfu Zhou, Chaoyue Li, Xi Liu, Jialiang Pei, Yifan Zhu, Liang Liu, and Huadong Ma. Reviving peer-to-peer networking for scalable crowdsourced live video streaming. *IEEE/ACM TON*, 2024.

[64] Konstantin Andreev, Bruce M Maggs, Adam Meyerson, and Ramesh K Sitaraman. Designing overlay multicast networks for streaming. In *ACM symposium on Parallel algorithms and architectures*, 2003.

[65] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. C3:{internet-scale} control plane for video quality optimization. In *USENIX NSDI*, 2015.