This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

BufTune: Reducing Traffic Waste via Seek-Aware Buffer Tuning in Interactive Video Streaming

Lihuan Hui, *Student Member, IEEE,* Weifeng Fan, Wang Yang, *Member, IEEE,* Fan Wu, *Member, IEEE,* Yanbo Wang, *Student Member, IEEE,* Feng Lyu, *Senior Member, IEEE,* and Yaoxue Zhang, *Senior Member, IEEE*

***Abstract*—In modern Video-on-Demand (VOD) services, non-linear viewing behaviors such as seeking and skipping are increasingly common. However, current streaming systems often rely on aggressive prefetching and large fixed playback buffers to ensure smooth playback, which can result in substantial traffic waste when prefetched segments are discarded due to frequent user interaction. To mitigate this inefficiency, we propose *BufTune*, a seek-aware client-side buffer tuning strategy. *BufTune* dynamically adjusts the playback buffer size in response to user seek behavior—shrinking the buffer during frequent interactions to avoid downloading segments unlikely to be watched, and gradually restoring it during stable playback to maintain robustness, thereby mitigating traffic waste while preserving, or even improving, QoE. *BufTune* comprises two core modules: a *User Behavior Detector* that detects seek events and coordinates playback adaptation, and a *Buffer Tuner* that adjusts the buffer size—shrinking it during frequent seeks to reduce unnecessary prefetching, and progressively restoring it during stable playback to ensure smooth viewing experience. Experimental results across multiple ABR algorithms and network conditions demonstrate that, compared with the fixed buffer approach, *BufTune* consistently reduces traffic waste by up to 38.5%, while delivering QoE improvements of up to 4.9%. These results highlight *BufTune*'s effectiveness and generality in addressing the emerging challenges of interactive VOD streaming.**

***Index Terms*—Buffer Tuning, Video Streaming, Scalable Video Coding (SVC), Adaption Bitrate (ABR)**

I. Introduction

In recent years, video streaming has rapidly emerged as a dominant mode of online content delivery, reshaping consumption behavior and accounting for a significant share of global internet traffic [1]–[5]. Among various streaming formats, long-form Video-on-Demand (VOD) content [6]–[9], as delivered by platforms such as Netflix and YouTube, is particularly prevalent. According to Sandvine's Global Internet Phenomena Report 2024 [10], VOD streaming constitutes 57% of global mobile traffic, underscoring VOD's significance in the streaming ecosystem.

VOD sessions, unlike live streaming or short-form video, increasingly exhibit interactive playback behaviors [11]–[16]. A key aspect of this interactivity is non-linear seeking, which includes skipping intros/outros, fast-forwarding through uninteresting segments, and jumping to points of interest [13], [17]–[19]. These behaviors introduce new challenges for client-side adaptation strategies, particularly for Scalable Video Coding (SVC), where video content is encoded into a base layer (BL) and multiple enhancement layers (ELs) [20], [21]. SVC allows clients to flexibly adapt video quality based on real-time bandwidth conditions, making it well-suited for fluctuating network environments [22]–[24]. In practice, SVC is often deployed within Dynamic Adaptive Streaming over HTTP (DASH) systems—forming DASH-SVC [25], [26], which supports layer-aware streaming by enabling clients to fetch only the necessary base and enhancement layers.

To maintain high QoE, recent DASH-SVC adaptive bitrate (ABR) algorithms [27]–[31] tend to adopt aggressive prefetching strategies, selecting higher bitrates and proactively downloading more ELs in anticipation of smooth playback. While effective during continuous playback, this prefetching strategy becomes problematic in interactive VOD scenarios with frequent seeking. When users jump across the video timeline, previously downloaded segments are often discarded before being played, resulting in substantial traffic waste. *The more aggressively ABR algorithms favor higher bitrates, the more enhancement layers are prefetched to maintain QoE; and the more frequently users seek, the more likely these prefetched segments are to be discarded, resulting in substantial traffic waste and increased network load.* Ultimately, this tension reflects a core conflict between QoE optimization and traffic efficiency in interactive VOD streaming [18], [32]–[34].

This raises a central question: *Does improving traffic efficiency necessarily come at the cost of QoE? Can we retain the QoE benefits of existing SVC ABR algorithms while mitigating traffic waste caused by interactive playback?* While several prior studies have explored ways to reduce redundancy in video delivery—such as detecting skippable segments [13] or modeling playback patterns [11], [18], [35]—these approaches largely operate at the content or delivery layer. Other works seek to improve bandwidth efficiency by refining ABR logic [36]–[39], typically by optimizing bitrate selection or prefetch thresholds based on network and playback patterns. In contrast, the adaptation of playback buffer behavior in response to user seeking remains underexplored, particularly within the context of SVC-based adaptive streaming.

We propose that one potential cause of this inefficiency is the widespread reliance on fixed, large playback buffers in existing SVC-based ABR algorithms. While such buffers provide resilience against short-term network fluctuations,

Lihuan Hui, Weifeng Fan, Wang Yang, Fan Wu, Yanbo Wang, and Feng Lyu are with the School of Computer Science and Engineering, Central South University, Changsha, 410083, China (email:{lh_hui,224711016,yangwang,wfwufan,wangyb,fenglyu}@csu.edu.cn)

Yaoxue Zhang is with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, 100084, China, and also with Zhongguancun Laboratory, Beijing,100094, China (email: zhangyx@tsinghua.edu.cn).

(Corresponding author: Wang Yang.)

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

they also increase the risk of prefetching high-bitrate content that is never played, especially when playback is frequently interrupted by user seeking behaviors.

A seemingly intuitive solution is to simply shrink the playback buffer, thereby limiting speculative prefetching and lowering the risk of traffic waste. However, this introduces a new problem: small buffers reduce resilience to short-term bandwidth fluctuations, significantly increasing the likelihood of playback stalls, one of the most damaging factors to QoE.

Motivated by this, we propose *BufTune*—a seek-aware buffer tuning strategy that dynamically adjusts the client-side playback buffer in response to non-linear seeking behavior. When frequent seeking is detected, *BufTune* reduces the buffer size to limit speculative downloads and redirect bandwidth to segments with higher playback probability, namely those temporally closer to the current playback position, as they are more likely to be viewed in the near future. As playback stabilizes, the buffer is gradually restored to avoid rebuffering.

To achieve this, *BufTune* adopts a modular design with two key components. The *User Behavior Detector* continuously monitors playback to detect seeking actions and triggers system-wide responses, instructing the video player to jump to the new playback position, notifying the ABR logic to cancel in-flight requests and fetch segments based on the updated timeline, and signaling the *Buffer Tuner* to reassess the buffer size. The *Buffer Tuner* governs buffer tunning through two complementary controllers: a *Shrink Controller*, which reduces the buffer size in response to frequent seeking to avoid speculative downloads, and an *Enlarge Controller*, which progressively restores the buffer during stable playback to maintain uninterrupted playback. Working in tandem, these components allow *BufTune* to respond timely to user behavior and mitigate traffic waste without compromising QoE.

The contributions are detailed as follows:

- We identify and characterize the problem of traffic waste in interactive VOD streaming, stemming from the mismatch between aggressive prefetching and frequent seeking behaviors. Empirical analysis shows that frequent seeking leads to substantial amounts of downloaded data being discarded without playback.
- We propose *BufTune*, a client-side buffer tuning mechanism that dynamically adapts to user behavior. It consists of two key components: a *User Behavior Detector* that detects seek events and coordinates system-level adaptation, and a *Buffer Tuner* that adaptively resizes the buffer—shrinking it during frequent interactions to limit unnecessary prefetching, and gradually restoring it as playback stabilizes to maintain robustness.
- *BufTune* is implemented as a client-side-only, ABR-agnostic module that requires no server modifications, making it broadly compatible and easily deployable in streaming platforms.
- We implement *BufTune* on a research-grade DASH client (AStream) and evaluate it under a range of network conditions emulated via Mininet. Designed to reduce traffic waste without compromising the QoE benefits of existing SVC ABR algorithms, *BufTune* reduces traffic waste by up to 38.5% and improves QoE by up to 4.9% across multiple ABR schemes. These results demonstrate its effectiveness and generality in interactive VOD scenarios.

The rest of this paper is organized as follows: Section II presents background and motivation for seek-aware buffer tuning. Section III presents the design of *BufTune*. Section IV evaluates the effectiveness and generality of *BufTune* using Mininet. Section V reviews related work. Finally, Section VII concludes the paper.

II. Background and Motivation

This section provides background on interactive VOD behavior and DASH-SVC ABR algorithms, then presents empirical evidence showing how frequent seeking can lead to traffic waste. We further analyze the root cause of this inefficiency and motivate the need for seek-aware buffer tuning.

A. Background

Rise of Interactive VOD Behavior. Modern VOD platforms are characterized by a high degree of interactivity [12]–[14]. Users routinely engage in seeking behaviors, such as scrubbing along the timeline [15], and skipping intros/outros [16], particularly in long-form content. These non-linear behaviors are now embedded in everyday viewing and have gradually shifted user engagement away from passive, start-to-finish consumption toward personalized and selective exploration. To support this shift, Platforms like Netflix [40] and YouTube [41] have embedded features such as “skip intro,” auto-chapters, and interactive previews [42]. These design choices reflect a broader shift in expectation: users want granular control over what and how they watch.

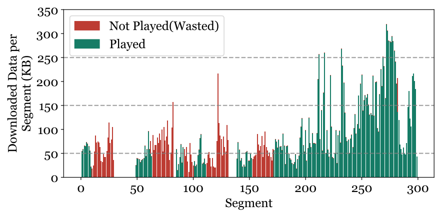
However, despite the prevalence of such behaviors, many underlying system mechanisms, including ABR logic, remain optimized for uninterrupted, linear playback, often relying on aggressive prefetching strategies assuming continuous consumption [27]–[30], [43]–[46]. While front-end features increasingly support interactive usage, backend systems have largely retained assumptions rooted in linear viewing. These assumptions stem from traditional streaming architectures, which are built around stable, forward-only playback and rely on large fixed-size buffers and speculative prefetching to ensure smooth viewing. However, in interactive VOD contexts, these assumptions often break: the system may prefetch segments that are quickly skipped, leading to unnecessary traffic without any improvement in user-perceived quality.

ABR Algorithms in SVC-based Streaming. Advanced Video Coding (AVC) [20] is a common encoding standard that encodes video at discrete quality levels. SVC [47] extends AVC by introducing a layered structure, where each segment consists of a base layer (BL) and several enhancement layers (EL). The BL provides the minimum playback quality, while ELs can be added incrementally to improve resolution [48]. This layered structure enables fine-grained quality adaptation under varying network conditions [27].

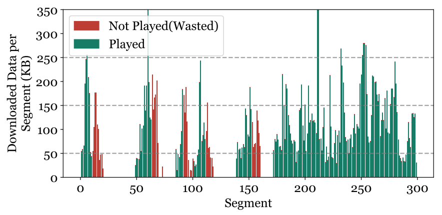
In SVC, adaptive bitrate (ABR) algorithms determine which video segments—and more specifically, which layers—should be requested at runtime based on current throughput estimates and buffer status. Unlike conventional approaches that make

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

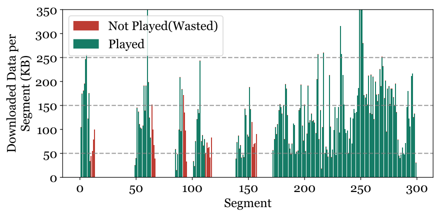
This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870



(a) *Backfilling* with 5 Seeks

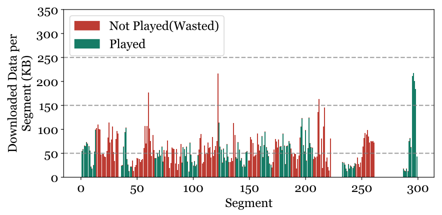


(b) *mpBack*^{2} with 5 Seeks

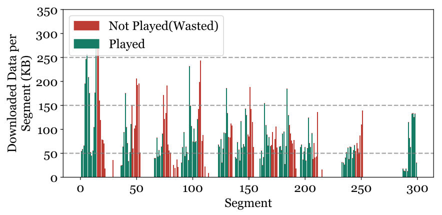


(c) *RASD* with 5 Seeks

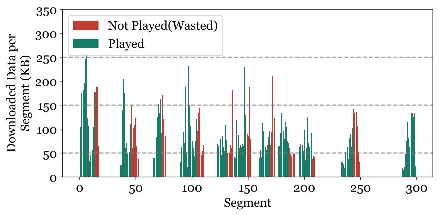
Fig. 1: Segment-level playback behavior of three SVC ABR algorithms with 5 Seeks: Played vs. Wasted Data. *Green bars indicate segments that were downloaded and played; red bars represent segments that were downloaded but ultimately wasted.*



(a) *Backfilling* with 10 Seeks



(b) *mpBack*^{2} with 10 Seeks



(c) *RASD* with 10 Seeks

Fig. 2: Segment-level playback behavior of three SVC ABR algorithms with 10 Seeks: Played vs. Wasted Data. *More frequent seeking leads to visibly greater traffic waste, reflecting the growing inefficiency under interactive playback.*

coarse-grained decisions at the segment level, ABR in SVC must operate at a finer granularity. The client can initiate playback immediately with only the BL, and selectively request additional ELs as network conditions permit. This enables more responsive and efficient adaptation to bandwidth fluctuations, while maintaining continuous playback.

To improve the playback QoE under this framework, a variety of SVC ABR algorithms have been proposed [27]–[29], [43], [45], [46], [49], [50]. These can be roughly grouped by their adaptation focus. *Buffer-based* methods like *Backfilling* [27] prioritize keeping a full buffer of BLs, then opportunistically backfill ELs in reverse. *Hybrid* methods like *RASD* [28] dynamically decide whether to fetch additional enhancement layers after each base layer download, depending on buffer status and recent bandwidth estimates. Other methods consider multipath delivery contexts—for example, *mpBack*^{2} [29] schedules different video layers across multiple network interfaces and adjusts bitrate selection based on per-path bandwidth availability. Beyond these, additional designs target segment size prediction [46], network architectures like NDN [45], or cooperative group scenarios [50].

These approaches reflect a growing effort to exploit SVC's scalability for better QoE. However, they primarily target stable or forward-only playback settings, with limited attention to dynamic behavioral patterns such as frequent seeking.

B. Motivation

Limitations of SVC ABR in Interactive Scenarios. While effective under linear playback, is aggressive prefetching by SVC ABR algorithms still beneficial when users frequently seek through a video? These algorithms aim to improve QoE by downloading higher-bitrate segments in advance, but such prefetched content may be discarded entirely when users skip or jump across the timeline. Intuitively, the higher the target quality, the more data is fetched—and the more often the user seeks, the more of that data may be wasted.

To validate this hypothesis, we conduct a Mininet-based [51] emulation using the *Big Buck Bunny* [52] (10 minutes, 720p, 2-second segments). The client and server are connected via a two-path topology, and network conditions are dynamically varied using Pensieve's network traces [53] via the Linux `tc` tool. During playback, we inject 5 and 10 randomly positioned seek operations to simulate interactive behavior.

We evaluate three representative DASH-SVC ABR algorithms: *Backfilling* [27], *RASD* [28], and *mpBack*^{2} [29]—and visualize their segment-level playback behavior by recording which segments are downloaded and whether they are eventually played or discarded, as shown in Fig. 1 and Fig. 2, respectively. In these visualizations, green bars indicate segments that were downloaded and played, while red bars represent segments that were downloaded but skipped—i.e., wasted. We make three key observations. First, all three ABR algorithms incur substantial traffic waste, even under moderate levels of interactivity. Second, the proportion of wasted data increases significantly as the frequency of seek operations rises. Third, many of the wasted segments appear near the tail of the buffer and are fetched at relatively high quality layers, as reflected by the increased concentration of red segments toward the end of the buffer. These results confirm our hypothesis that frequent interactivity amplifies traffic waste.

The Need for QoE-Preserving Waste Mitigation. Historically, VOD providers reduced bandwidth waste through network-side techniques such as TCP window-based traffic shaping [54], [55] and Deep Packet Inspection (DPI)-based

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

flow classification [56], motivated by pricing models tied to peak traffic usage. These methods effectively limited aggregate bandwidth usage but did not consider playback QoE.

Today, adaptive video delivery has shifted toward client-side control, where ABR algorithms make prefetching and bitrate decisions to optimize playback QoE. Meanwhile, the widespread adoption of Hypertext Transfer Protocol Secure (HTTPS), Quick UDP Internet Connections (QUIC), and Hypertext Transfer Protocol Version 3 (HTTP/3) has made video traffic fully encrypted and encapsulated over User Datagram Protocol (UDP). This shift significantly reduces the visibility and controllability of in-network devices [57], [58]. Moreover, network-side shaping can conflict with client-side ABR logic, leading to playback instability and QoE degradation [59]. Taken together, these developments render in-network traffic shaping ineffective in practice and naturally lead to the question: *Can we preserve the high QoE of modern ABR algorithms while reducing unnecessary waste under interactive playback?*

Toward Seek-Aware Buffer Tuning. To understand the root of traffic waste, we revisit how SVC ABR algorithms operate in DASH-based systems. Most DASH-SVC systems rely on fixed, relatively large playback buffers to ensure smooth viewing in the face of fluctuating network conditions. These buffers are filled aggressively, with the ABR logic continually probing available bandwidth and prefetching segments—often at high quality layers—to maximize visual quality. However, when a seek operation occurs, these prefetched segments may no longer be needed, resulting in wasted traffic. The larger the buffer, the further ahead the ABR decisions are made, and the more data is at risk of being skipped. Thus, while large buffers improve resilience under linear playback, they also amplify bandwidth waste under interactive usage.

This observation highlights a key insight: *the problem does not lie in ABR itself, but in how far ahead its decisions are projected*. If we could dynamically limit this scope by reducing the prefetch range when user interactions are frequent, we might retain ABR's QoE benefits while reducing waste. This forms the core idea behind *BufTune*: a seek-aware buffer tuning mechanism that adapts buffer size based on user behavior.

III. BufTune Design

Building on the insights discussed above, we design *BufTune*, a seek-aware buffer tuning mechanism tailored for SVC-based adaptive streaming under interactive VOD scenarios, shown in Fig. 3. In this section, we first present an overview and workflow of *BufTune* (§III-A), followed by detailed explanations of its core components: the *User Behavior Detector* (§III-B) and the *Buffer Tuner* (§III-C).

A. BufTune's Overview and Workflow

BufTune dynamically adjusts the playback buffer size in response to user seeking behavior, aiming to reduce traffic waste without sacrificing playback QoE. When frequent seeking is detected, it reduces the buffer size to avoid downloading high-bitrate segments that are unlikely to be viewed; when playback is stable, it gradually enlarges the buffer to maintain robustness. This behavior is coordinated by two core components:

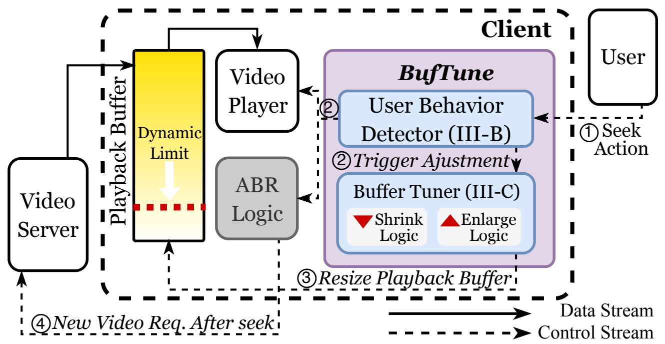


Fig. 3: Architecture of *BufTune*. *Red dashed line within the playback buffer indicates the temporarily reduced buffer capacity, as determined by BufTune in response to seek behavior.*

the *User Behavior Detector*, which detects seeking actions and triggers system-wide responses, and the *Buffer Tuner*, which determines buffer resizing actions using shrink and enlarge logic. Integrated with the ABR logic and video player, this design allows it to be both responsive to user behavior and agnostic to specific ABR implementations.

Workflow. The workflow of *BufTune* is as follows:

- $\mathcal{S}1$**: Seek Action Detection.** When the *User Behavior Detector* detects a user-initiated seeking operation during playback, it coordinates actions across modules: it instructs the *Video Player* to jump to the target position, alerts the *Buffer Tuner* to reassess buffer size, and signals the ABR Logic to cancel in-flight requests and issue a new video segment request based on the updated position. It also sends a reset timestamp to the *Buffer Tuner* to mark the start of a new playback phase.
- $\mathcal{S}2$**: Trigger Adjustment.** Upon receiving the timestamp reset by the *User Behavior Detector*, the *Buffer Tuner* analyzes seek frequency within a sliding time window. Each seek event triggers the *Shrink Logic* to reduce the playback buffer limit, with the reduction magnitude determined by recent seek frequency. Meanwhile, the timestamp serves as a reference for the *Enlarge Logic*, which gradually increases the buffer size based on the duration of uninterrupted playback.
- $\mathcal{S}3$**: Buffer Reconfiguration.** The *Buffer Tuner* enforces the new buffer limit by updating playback parameters, which constrain how far ahead segments can be prefetched. This configuration is synchronized with the playback state to maintain consistency.
- $\mathcal{S}4$**: Segment Request Renewal.** Guided by the updated playback position and buffer limit, the ABR Logic reissues a video segment request. This process continues until the next seek event triggers a new adjustment cycle or playback completes.

Together, these steps enable *BufTune* to adaptively manage the playback buffer size under interactive conditions, effectively reducing traffic waste without compromising QoE.

Importantly, while *BufTune* does not alter ABR logic itself, it influences bitrate decisions by dynamically adjusting buffer size in response to user behavior. A smaller buffer reduces speculative prefetching, allowing ABR to reallocate bandwidth

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

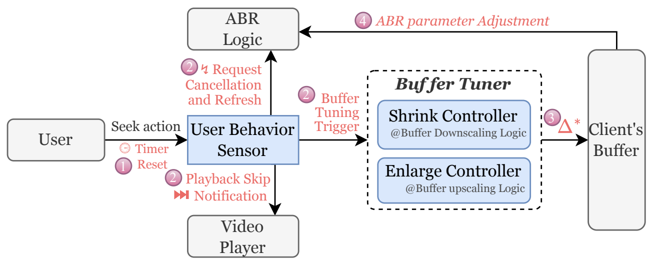


Fig. 4: System response workflow to seek behaviors in *BufTune. Upon detecting a seek, the User Behavior Detector coordinates playback skip, request cancellation, and buffer tuning trigger.*

more promptly toward enhancement layer fetching within a narrower playback buffer. Once the buffer is fully filled, downloads pause, and the remaining bandwidth is preserved, which achieves both improved playback quality and reduced traffic waste. This design enables *BufTune* to complement existing SVC ABR schemes without modifying their internal mechanisms. We now detail the design of each component.

B. User Behavior Detector

The *User Behavior Detector* serves as the system's central coordination point for responding to user interactions, particularly seeking. While seek behavior can be passively detected via standard media player APIs (e.g., changes in playback position), the *User Behavior Detector* explicitly orchestrates multiple downstream adaptations in response to these events.

Upon detecting a seek action during video playback, the Detector performs four parallel actions (see Fig. 4). First, it resets a global timestamp—the *enlarge timer*—later used by the *Enlarge Logic* to estimate how long playback has remained uninterrupted. Simultaneously, it triggers three concurrent responses: (1) instructs the *Video Player* to jump to the target playback position (*Playback Skip Notification*); (2) signals the *ABR Logic* to cancel in-flight segment requests and issue new ones based on the updated timeline. Specifically, it terminates the ongoing segment request by sending a `CANCEL_FRAME` to the server, thereby preventing unnecessary resource usage caused by behavioral changes (*Request Cancellation and Refresh*); and (3) forwards the *enlarge timer* to the *Buffer Tuner*. The timestamp serves as input to both the *Shrink* and *Enlarge* logic: the former uses it to assess seek frequency within a sliding time window, while the latter uses it as a reference to track uninterrupted playback (*Buffer Tuning Trigger*).

This tightly coupled coordination ensures that buffer adaptation is both prompt and responsive to interactive user behavior. We next detail the internal design of the *Buffer Tuner*.

C. Buffer Tuner

The *Buffer Tuner* is responsible for promptly and dynamically adjusting the client-side playback buffer in response to user interaction patterns. Its primary goal is to balance playback robustness and bandwidth efficiency by shrinking the buffer to avoid unnecessary prefetching when seeking is frequent, and gradually restoring it as playback stabilizes. To

TABLE I: The Key Notations Used in the Paper

Notation	Description
B_{current}	Current playback buffer size (in segments)
B_{init}	Initial buffer size before any shrink action
$B_{\min}$	Minimum buffer size to ensure uninterrupted playback
B_{target}	New buffer size after shrink adjustment
ts	List of timestamps for recent seek events
L	Sliding time window length (e.g., 60s)
t_{seek}	Timestamp of current seek action
$\mathcal{N}_{\text{seek}}$	Number of seek events in the sliding window
β	Decay aggressiveness coefficient; Recommended Range: 0.1–0.4 (default: 0.3)
$inBuffer$	Boolean flag: true if seek target is in buffer
Δ	Calculated buffer enlargement increment
ξ	Scaling coefficient for bandwidth surplus; Recommended Range: 0.4–0.5 (default: 0.5)
δ	Base increment factor controlling minimum growth per step; Recommended Range: 0.2–0.5 (default: 0.3)
BW_{highest}	Highest video bitrate available
BW_{base}	Bitrate of the base layer
BW_{pred}	Predicted available bandwidth

achieve this, the *Buffer Tuner* employs two complementary components: the *Shrink Controller*, which implements buffer downscaling logic, and the *Enlarge Controller*, which implements buffer upscaling logic, as illustrated in Fig. 4. Both components share a common input—the *enlarge timer*—which tracks playback stability and is reset on each seek event.

At the core of *Buffer Tuner* lies a dual-adaptation mechanism. The *Shrink Controller* monitors seek frequency over a sliding time window and selectively reduces the buffer size upon out-of-buffer seeks, with the reduction magnitude proportional to recent seek frequency. In contrast, the *Enlarge Controller* incrementally increases the buffer during uninterrupted playback, with the pace of enlargement determined by playback stability. Together, these mechanisms enable *BufTune* to remain responsive to user interaction while maintaining overall playback robustness. Building on this dual-mechanism framework, we next examine the specific roles and algorithms of the *Shrink Controller* and *Enlarge Controller*.

1) Shrink Controller – Buffer-Aware and Frequency-Sensitive Logic: To mitigate traffic waste due to seek behavior, the *Shrink Controller* uses a dual-factor shrink strategy that considers both the spatial position of seek targets and the temporal frequency of seek events. This design ensures that buffer reduction is only triggered when it is likely to reduce actual traffic overhead and scales with recent interaction intensity.

Buffer-Aware Triggering: To avoid overreacting to non-disruptive user behaviors, buffer adaptation is conditionally triggered only when a seek operation is expected to result in redundant data transfer. This determination is made by assessing the spatial locality of the seek—that is, whether the target segment lies within the current playback buffer:

- **In-Buffer Seek:** If the target segment is already present in the buffer, no additional data fetching is needed. In this case, buffer shrinking is skipped to avoid unnecessary adaptation. The *Enlarge Timer* may be unaffected. (See "Seek Target: 1" in Fig. 5.)
- **Out-of-Buffer Seek:** If the target lies outside the buffer, shrinking is triggered to constrain unnecessary prefetching, thereby reducing the likelihood of traffic waste.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

The degree of buffer reduction is further adjusted based on recent seek frequency, as detailed in the frequency-sensitive adaptation logic discussed next. (See "Seek Target: 2" in Fig. 5.)

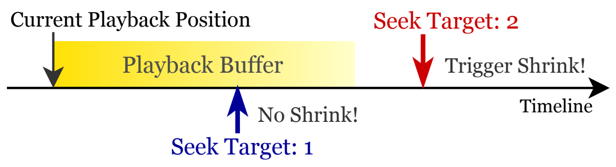


Fig. 5: Handling of In-Buffer vs. Out-of-Buffer Seeks. *Only out-of-buffer seeks trigger buffer resizing via the shrink controller; in-buffer seeks bypass adaptation logic. Shrink decisions are governed by Frequency-Sensitive Adaptation.*

Frequency-Sensitive Adaptation: To ensure that buffer adaptation reflects user interaction intensity, the *Shrink Controller* incorporates a frequency-sensitive adaptation strategy that dynamically scales the aggressiveness of buffer reduction based on the number of seek events within a sliding time window. A higher seek frequency indicates increased playback disruption and a greater likelihood of traffic waste, prompting more aggressive buffer shrinking. Conversely, infrequent seeks yield gentler adjustments to preserve playback continuity.

When shrink logic is activated, the updated buffer limit B_{target} is computed as:

$$B_{\text{target}} = \max(B_{\min}, B_{\text{current}} \cdot e^{-\beta \cdot \mathcal{N}_{\text{seek}}}), \tag{1}$$

where B_{current} is the current buffer size (equal to the initial buffer size B_{init} on the first decision), $B_{\min}$ is the minimum allowed buffer size to ensure playback continuity, $\mathcal{N}_{\text{seek}}$ is the number of seek operations within a sliding time window, and β is a decay coefficient that controls how aggressively the buffer should be reduced in response to frequent seeks (typically set between 0.1 and 0.4, with a default value of 0.3).

This approach enables *BufTune* to mitigate traffic waste in a behavior-aware manner while preserving playback resilience. To operationalize the buffer-aware and frequency-sensitive shrink strategy described above, we implement the logic in Algorithm 1. Upon detecting a seek action, the algorithm first checks whether the target segment lies inside the current buffer. If the segment is already buffered, no shrink is performed (*line* 4∼6). Otherwise, the algorithm counts recent seek events within a sliding time window and calculates the target buffer limit using Eq. (1) (*line* 9∼14). Finally, it adjusts the ABR logic to conform to the new buffer size limit, ensuring consistent prefetching behavior (*line* 15).

This shrink algorithm forms a critical part of *BufTune*'s adaptive logic and works in tandem with the enlarge controller to balance responsiveness and playback resilience.

2) Enlarge Controller – Adaptive Enlargement Logic: The *Enlarge Controller* performs the inverse function of its shrink counterpart, gradually restoring the playback buffer size after playback stability is re-established following one or more seeks. It leverages the *enlarge timer*, which tracks uninterrupted playback duration since the last seek event, to determine when and how fast to resume buffer growth. Its primary objective is to recover from previous reductions quickly

Algorithm 1: *Buffer-Aware Frequency-Sensitive Shrink Algorithm*

```
Input:
  $ts$: list of recent seek timestamps;
  $B_{init}$: initial buffer size;
  $B_{\min}$: minimum buffer threshold;
  $inBuffer$: boolean indicating whether the seek target
  lies within the current buffer.
Output: $B_{\text{target}}$: Updated buffer limit.
1  $ts \leftarrow []$
2  $B_{\text{current}} \leftarrow B_{init}$
3  if the User Behavior Detector detects a seek event then
4      if $inBuffer$ then
5          $B_{\text{target}} \leftarrow B_{\text{current}}$
6      end
7      else
8          $now \leftarrow$ current system time
9          $ts$.append($now$)
10         while $t_{seek} - ts[0] > L$ do
11             $ts \leftarrow ts[1:]$        // Remove outdated seeks
12         end
13         $N_{\text{seek}} \leftarrow \text{len}(ts)$        // Seek count in sliding window
14         $B_{\text{target}} \leftarrow \max(B_{\min},\ B_{init} \cdot e^{-\beta \cdot N_{\text{seek}}})$
15         Adjust ABR to new buffer limit   // apply new threshold
16     end
17 end
18 return $B_{\text{target}}$
```

enough to enhance QoE, while avoiding overly aggressive growth that may lead to unnecessary traffic consumption.

To achieve this, the controller implements an adaptive enlargement strategy that scales the buffer increment Δ based on two key factors: playback stability and bandwidth availability.

Step Factor: Playback Stability Driven Scaling: The first component, a time-based discrete step factor, reflects increasing confidence in playback continuity:

$$\text{StepFactor}(T_{\text{stable}}) = \begin{cases} 1, & 0 < T_{\text{stable}} \leq 10 \\ 2, & 10 < T_{\text{stable}} \leq 20 \\ 3, & T_{\text{stable}} > 20 \end{cases} \tag{2}$$

Bandwidth-Aware Adjustment: To further adapt to network conditions, the actual buffer increment $\Delta(t)$ incorporates available bandwidth headroom:

$$\begin{aligned} \Delta = {} & \text{StepFactor}(T_{\text{stable}}) \\ & \cdot \left(\xi \cdot \max\left(\frac{BW_{\text{highest}} - BW_{\text{pred}}}{BW_{\text{base}}},\ 0 \right) + \delta \right), \end{aligned} \tag{3}$$

where BW_{highest} is the highest video bitrate available, BW_{pred} is the predicted network bandwidth, and BW_{base} is the base layer bitrate. The constants ξ and δ control the influence of bandwidth surplus and guarantee a minimum growth step, respectively. BW_{pred} is directly inherited from the underlying

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

Algorithm 2: Incremental Buffer Recovery Algorithm

```
Input: Periodic timer interval $T$ (set to half of the
       current buffer size)
1  // Continuous incremental recovery
   during steady-state playback
2  while True do
       // Start the buffer growth timer
3      $timer \leftarrow$ initialize timer($T$)
4      if timer is triggered then
           // Compute dynamic step size
5          $\Delta \leftarrow \frac{\max(BW_{\text{highest}} - BW_{\text{pred}},\ 0)}{BW_{\text{base}}} + 1$
           // Incremental buffer update
6          $B_{\text{current}} \leftarrow \min(B_{\max},\ B_{\text{current}} + \Delta)$
7          AdjustABRParameters()
           // Update ABR-related buffer
           constants
8      end
9  end
```

ABR algorithm. Specifically, most ABR schemes rely on bandwidth prediction to select an appropriate bitrate—higher predicted throughput allows for higher-quality selection, and vice versa. This shared signal ensures alignment between bitrate and buffer control without redundant prediction logic.

When the available bandwidth is high, the player can maintain stable playback, and thus buffer growth is conservative to prevent future traffic waste due to sudden seek operations. This also encourages the ABR algorithm to download enhancement layers earlier to improve video quality. Conversely, under low bandwidth conditions, the system prioritizes rapid buffer expansion to stabilize playback by filling BLs.

Buffer Size Update Rule: The target buffer limit is then computed as:

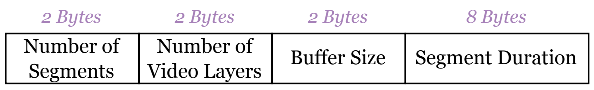
$$B_{\text{target}} = \min(B_{\text{init}},\ B_{\text{current}} + \lfloor \Delta(t) \rfloor), \tag{4}$$

where B_{current} is the current buffer size, B_{init} is the initial buffer size before shrink, and $\lfloor \cdot \rfloor$ denotes the floor function to ensure integer segment counts.

The logic of the enlarge controller is detailed in Algorithm 2. This design ensures that the buffer is restored cautiously at first, then accelerates as playback stabilizes and bandwidth allows. By limiting growth to the initial buffer size and adapting to real-time network conditions, *BufTune* enables fast recovery of playback robustness after user-initiated seeks, while maintaining efficient use of bandwidth by avoiding unnecessary prefetching.

3) ABR Parameter Adjustment: This subsection introduces enhancements to the ABR algorithms to support client-side customization features such as video seeking and buffer adjustment. To this end, two control frames are added: one for video initialization and another for synchronizing client playback configuration changes. The structure of the `VIDEO_INITIAL_FRAME` is shown in Fig. 6.

The purpose of the `VIDEO_INITIAL_FRAME` is to configure client-side playback settings at the start of video playback. It includes the following fields:

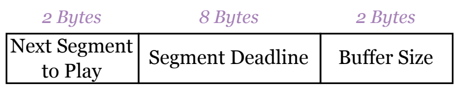


2 Bytes	2 Bytes	2 Bytes	8 Bytes
Number of Segments	Number of Video Layers	Buffer Size	Segment Duration

Fig. 6: Structure of the `VIDEO_INITIAL_FRAME`. All fields are encoded as unsigned integers.

1) *Number of Segments and Layers*: Two unsigned 16-bit integers indicating the total number of segments in the video and the number of layers per segment. This information facilitates the construction of a two-dimensional array to reduce the time and space overhead for video layer request lookup.
2) *Buffer Size*: An unsigned 16-bit integer that converts the server-side boundary parameter into a configurable client-side setting.
3) *Segment Duration*: An unsigned 64-bit integer representing the playback duration of each segment in milliseconds, used for deadline synchronization and related calculations.

The structure of the `CONF_SYNC_FRAME` (Configuration Synchronization Frame) is shown in Fig. 7.



2 Bytes	8 Bytes	2 Bytes
Next Segment to Play	Segment Deadline	Buffer Size

Fig. 7: Structure of the `CONF_SYNC_FRAME`.

The purpose of the configuration synchronization frame (`CONF_SYNC_FRAME`) is to notify the server of changes in playback configuration. It includes the following fields:

1) *Next Segment to Play*: An unsigned 16-bit integer indicating the next segment to be played. This is used to inform the server of playback progression or changes.
2) *Segment Deadline*: An unsigned 64-bit integer representing the Unix timestamp (in milliseconds) of the deadline for the upcoming segment.
3) *Buffer Size*: An unsigned 16-bit integer used to synchronize buffer configuration parameters with the server.

It is worth noting that the initialization frame is lightweight, occupying only 14 bytes, and is transmitted once per video session. The configuration synchronization frame is 12 bytes in size and is sent each time the client performs a seek operation or when the buffer size is adjusted. Compared to the time scale of video playback (measured in seconds), the communication overhead introduced by these control frames is negligible.

IV. Evaluation

We conduct comprehensive evaluations of *BufTune* using the Mininet [51], which enables the construction of realistic virtual topologies and the simulation of end-to-end video streaming workflows. The underlying network stack is configured with minRTT [60] as the packet scheduler and OLIA [61] as the multipath congestion control algorithm—both of which are standard modules in the Linux kernel. Besides, we implement the server in Go atop the MPQUIC protocol stack[1], and modify

[1]https://github.com/qdeconinck/mp-quic

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

the AStream player[2] to support SVC playback and dynamic buffer control.

A. Setup

Video sources. In our experiments, we use *Big Buck Bunny* [52] as the video source to evaluate the effectiveness of *BufTune*. The video is encoded using the SVC standard, with a resolution of $720p$ and a frame rate of $24fps$. Each stream includes one base layer (BL) and three enhancement layers (EL1∼EL3), and is segmented into 2-second chunks. The full duration is 10 minutes, which allows us to emulate typical user behaviors, including both continuous playback and frequent seeking. The average bitrates of the four layers are as follows: 189.84 KBps (BL), 330.80 KBps (EL1), 555.08 KBps (EL2), and 837.13 KBps (EL3). These values reflect the cumulative encoding bitrates up to each quality level, with the highest quality corresponding to the sum of all layers. Other SVC-encoded videos, including *Tears of Steel* [62] and *Elephants Dream* [63], are also evaluated in §IV-C4. Their average layer bitrates are listed in Table II.

TABLE II: Average bitrates of video layers in KBps

Video Source	BL	EL1	EL2	EL3
Big Buck Bunny (BBB)	189.84	330.80	555.08	837.13
Elephants Dream (ED)	202.76	344.37	596.53	1000.49
Tears of Steel (TOS)	183.37	316.65	539.98	885.64

Baselines. *BufTune* is designed to dynamically adjust the playback buffer size based on user seeking behavior, aiming to mitigate traffic waste. To evaluate both the effectiveness and generality of *BufTune*, we conduct a twofold comparison.

For *effectiveness*, we compare *BufTune* against a baseline approach with a fixed buffer size(e.g., 20 segments), where the buffer remains unchanged regardless of user seek behavior. We refer to this baseline as *FixedBuf*, representing the common default strategy used in most current SVC-based ABR implementations.

For *generality*, we integrate *BufTune* into three representative DASH-SVC ABR algorithms, namely *Backfilling* [27], *RASD* [28], and *mpBack*^{2} [29]. Below, we briefly describe each algorithm's original design.

(*i*) *Backfilling* [27]: is a buffer-based ABR algorithm that first fills the playback buffer with base layers in sequential order. Only after completing all base layer downloads does it start fetching enhancement layers in reverse segment order;

(*ii*) *RASD* [28]: is a hybrid ABR algorithm that, after downloading a base layer, evaluates buffer sufficiency and opportunistically backfills enhancement layers. These additional layers are selected with a conservative quality estimate, often below the level dictated by the current estimated bandwidth.

(*iii*) *mpBack*^{2} [29]: selects the current highest bitrate based on multipath bandwidth when the buffer is sufficient and uses the faster path's bandwidth when it is not; In *mpBack*^{2}, each subpath downloads video layer separately.

[2]https://github.com/pari685/AStream

Performance metrics. To evaluate the effectiveness of the proposed dynamic buffer adjustment strategy, *BufTune*, we adopt two key metrics: *traffic waste ratio* and *overall QoE*. Traffic waste rate quantifies the proportion of downloaded video data that is never played due to disruptive user behavior, and is defined as:

$$\eta_{\text{traffic\_waste}} = \frac{D_{\text{unplayed}}}{D_{\text{total}}} \times 100\%, \tag{5}$$

where D_{unplayed} denotes the amount of downloaded video data that was not actually played, and D_{total} is the total downloaded video data during playback. A lower value of $\eta_{\text{traffic\_waste}}$ indicates more efficient bandwidth utilization.

The overall QoE reflects the overall user experience and is computed based on five factors: the average *bitrate* during the playback session (aBr in KBps), *start-up delay* (sD), *quality switches times* (rC), count of stalling events (uC), and total *rebuffering time* during video playback (uD, in second). Following the formulation in *mpBack*^{2} [29], the overall QoE score is given by:

$$QoE = aBr \cdot a_0 - (sD \cdot a_1 + rC \cdot a_2 + uC \cdot a_3 + uD \cdot a_4), \tag{6}$$

where the coefficients are set to $a_0 = 0.236$, $a_1 = 0.049$, $a_2 = 0.092$, $a_3 = 0.436$, and $a_4 = 0.187$, consistent with *mpBack*^{2} [29]. All reported results are averaged over 50 runs.

Network traces. To evaluate *BufTune* under diverse and realistic network conditions, we constructed an experimental environment where a client retrieves video data from a server via two distinct network paths, configured with latencies of $22ms$ and $53ms$, respectively, and each with a packet loss rate of 1%. We utilized 20 publicly available network traces from the open-source Pensive dataset [53], which span a broad spectrum of bandwidth dynamics and variability, and randomly paired them to create 10 diverse test scenarios[3]. Each pair of traces was mapped to the two paths to emulate asymmetric and time-varying network conditions typical of real-world streaming environments. The aggregated downlink bandwidths of the 10 test scenarios are visualized in Fig. 8.

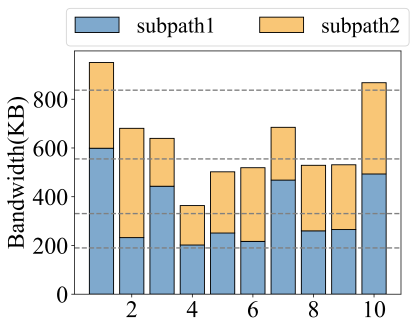


Fig. 8: Average downlink bandwidth in each test

B. Performance Evaluation of BufTune across Different ABR Algorithms

To evaluate the effectiveness of the *BufTune* across different SVC ABR algorithms, we conducted experiments using the

[3]Traces used in our analysis are available at: https://github.com/lh-hui/BufTune

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

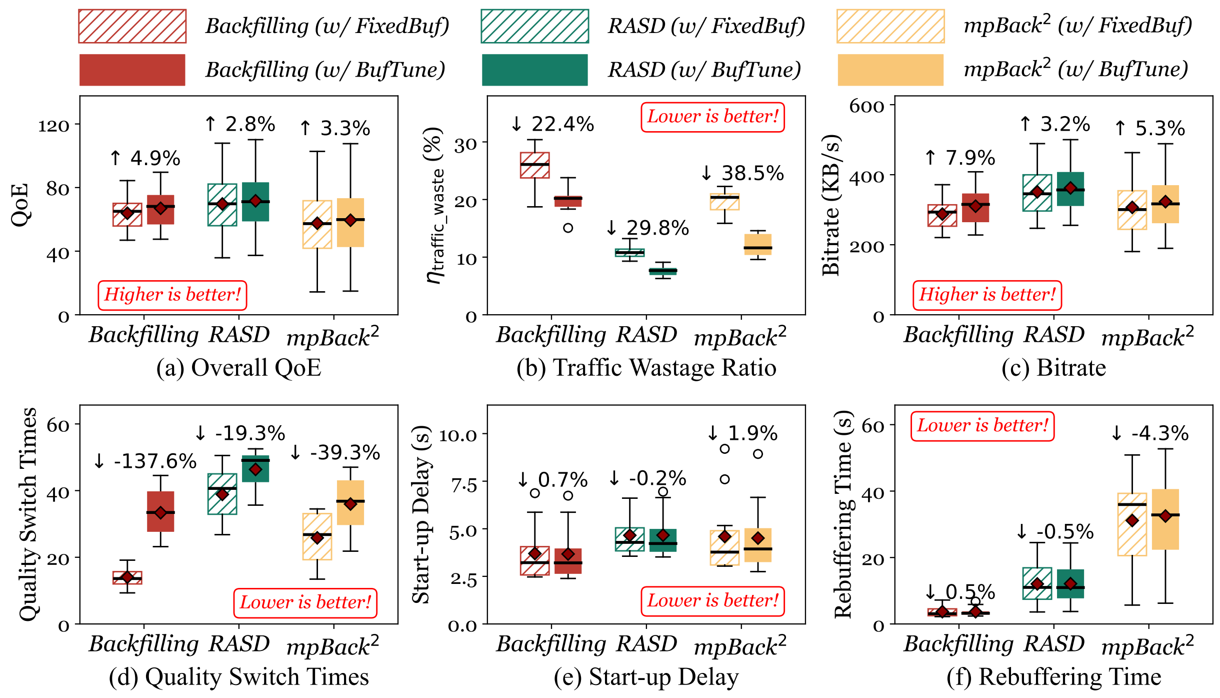


Fig. 9: Effect of *BufTune* across three SVC ABR Algorithms under Diverse Network Conditions. *Box plots show the median (horizontal line), interquartile range (box), and outliers (circles). Red diamonds indicate the mean value across all runs.*

Big Buck Bunny video under ten test scenarios. Each experiment involves five user seek events and reports the average result across ten randomized seek traces. As illustrated in Fig. 9, "(*w/ FixedBuf*)" refers to the baseline configuration where a fixed-size client buffer is used, while "(*w/ BufTune*)" denotes the same ABR algorithm augmented with *BufTune*'s dynamic buffer adjustment strategy.

Across all ABR variants, *BufTune* consistently alleviates traffic waste and improves QoE, as shown in Fig. 9(a) and Fig. 9(b). Specifically, *Backfilling* sees a 4.9% QoE improvement and 22.4% reduction in traffic waste; *RASD* achieves a 2.8% QoE gain and 29.8% less waste; *mpBack²* delivers a 3.3% QoE boost with the highest traffic savings of 38.5%. These gains stem from *BufTune*'s ability to adjust the client-side playback buffer in response to user behavior and network dynamics. By avoiding unnecessary data preloading during likely user seeking, *BufTune* reduces superfluous data transfers while maintaining smooth playback quality. These findings confirm that *BufTune* successfully achieves its intended goal—retaining the QoE benefits of SVC ABR algorithms while substantially reducing traffic waste during interactive playback.

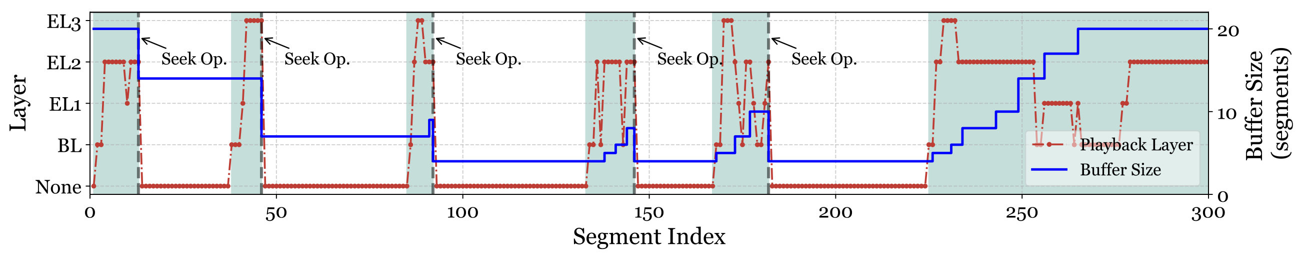
In addition to overall QoE and bandwidth waste, we further analyze how *BufTune* affects key QoE-related metrics, as shown in Fig. 9(c)–(f). We observe that *BufTune* generally improves the average *bitrate* across all three SVC ABR algorithms. This is because the buffer shrink triggered by seek events allows more bandwidth to be reallocated to enhancement layers, enabling higher-quality playback at the new video position. However, *BufTune* also leads to a noticeable increase in *quality switch times*. This is expected: after a seek, playback typically restarts from the base layer. Without buffer adjustment, the playback often remains at lower layers for longer, resulting in fewer switches. In contrast, *BufTune*'s shrink logic allows higher layers to be fetched more quickly within a smaller buffer window, leading to more frequent upward transitions. As illustrated in Fig. **??** Furthermore, *BufTune* does not significantly impact *start-up delay* or *rebuffering time*, indicating that bandwidth savings are achieved without sacrificing playback continuity.

To further visualize the data waste issue, we take *mpBack*^{2} as a case study under the first network scenario to illustrate how *BufTune* suppresses unnecessary data transfers during interactive playback. As shown in Fig. 11, downloaded-but-unplayed segments are marked in green, while segments that were played are shown in red. A comparison between Fig. 11(a) (without *BufTune*) and Fig. 11(b) (with *BufTune*) demonstrates that *BufTune* significantly reduces the amount of unplayed data. Moreover, it enables *mpBack*^{2} to more promptly increase the playback bitrate after each seek, thanks to adaptive buffer tuning. Similar behavioral patterns are also observed in the *Backfilling* and *RASD* algorithms (not shown here for brevity), indicating that *BufTune* provides consistent optimization benefits across different ABR strategies.

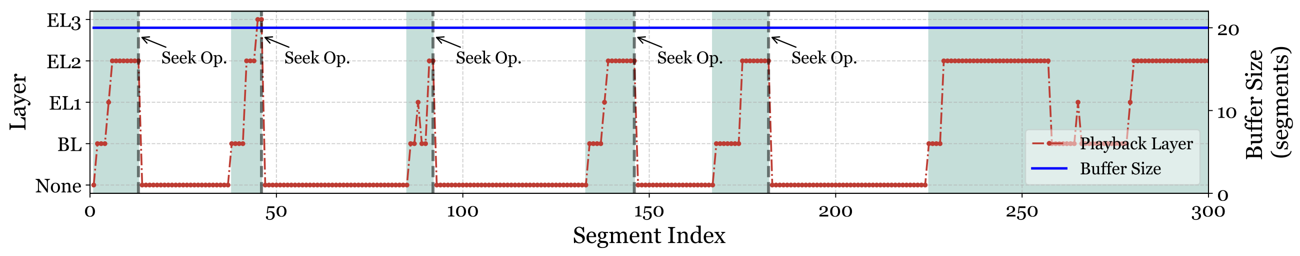
Practical Integration. To facilitate deployment, *BufTune* is designed as a lightweight module that works in tandem with existing DASH-SVC players. It runs alongside the ABR and reuses its bandwidth estimate, requiring no additional predictor. *BufTune* exposes two triggers: the buffer is shrunk only when a seek target lies outside the current buffer, and enlargement is evaluated once a fixed fraction (e.g., half) of the buffer has been played. Default parameters($\beta = 0.3$, $\xi = 0.5$, and $\delta = 0.3$) have proven robustness across ABR algorithms

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

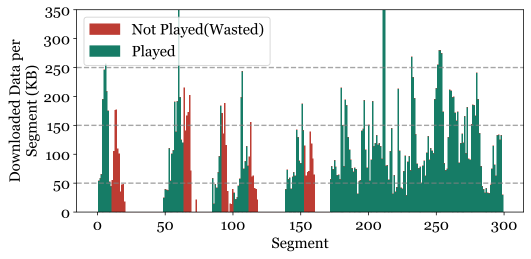


(a) With *BufTune*

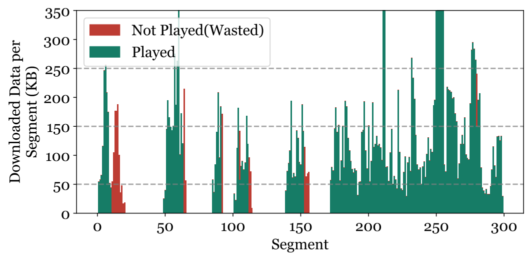


(b) With *FixedBuf*

Fig. 10: Controller Dynamics of $mpBack^2$: *BufTune* vs. *FixedBuf*. *Gray dashed lines indicate seek operations, green shaded areas mark played video intervals, red dashed curve shows the playback layer, and blue solid curve depicts the buffer level.*



(a) Playback Characteristics of $mpBack^2$(*w/ FixedBuf*)



(b) Playback Characteristics of $mpBack^2$(*w/ BufTune*)

Fig. 11: Video Layer Playback Behavior of $mpBack^2$ with and without *BufTune*.*Each row shows a video segment. "Played" segments were watched by the user; "Not Played" segments were downloaded but discarded due to seeking.*

and video contents. A full integration guide and tuning tips are provided at: https://github.com/lh-hui/BufTune.

C. Impact Evaluation

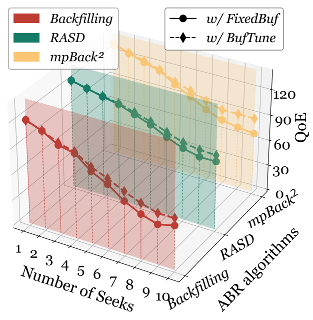
We conduct a series of impact studies to examine how key network conditions affect the performance of *BufTune*. Specifically, we analyze its effectiveness across varying network environments (seeking frequency, link delay, packet loss rate), multiple video contents, alternative fixed buffer baselines, and different hyperparameter configurations.

*1) **Impact of Seeking Frequency on BufTune**:* To evaluate how *BufTune* performs under different levels of video seeking frequency, we vary the seek frequency from 1 to 10 times per session. This range captures a spectrum from low to moderately high interactivity. One seek represents light interaction, while ten seeks—averaging one per minute—reflect frequent yet realistic user behavior. We avoid higher seek frequencies, as such behavior may indicate an impending user departure rather than continued interaction within the session. The results are shown in Fig. 12, where Fig. 12(a) illustrates QoE variation and Fig. 12(b) depicts the corresponding traffic waste ratio.

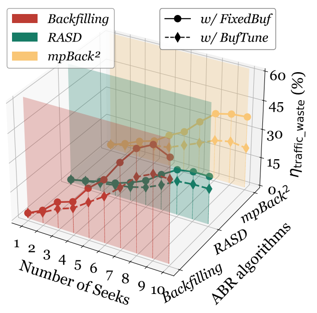
We observe two notable trends as the seek frequency increases. First, the overall QoE of all three ABR algorithms tends to degrade with more frequent seeking. However, the relative benefits of *BufTune* become increasingly pronounced. This is because more frequent jumps lead to more aggressive buffer shrinking, which in turn reallocates bandwidth toward higher enhancement layers within a smaller buffer range. Consequently, even in interaction-heavy scenarios, *BufTune* enhances perceptual quality by concentrating bandwidth on fewer, higher-quality segments. Second, as seek frequency increases, all algorithms show a steady rise in bandwidth waste. However, this increase slows down beyond 8 seeks. More importantly, the gap between '*w/ FixedBuf*' and '*w/ BufTune*' configurations further widens under high-frequency seeking, indicating stronger savings. This is due to two reasons: (*i*) *BufTune* applies more aggressive buffer shrinking when interaction is frequent, which suppresses unnecessary prefetching; (*ii*) When users seek too frequently in a 10-minute video, there is less time left for actual playback, naturally limiting further data waste. These two effects work together to amplify *BufTune*'s advantage under highly interactive conditions.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

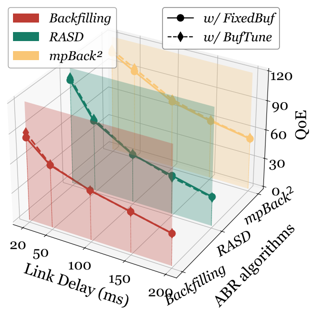


(a) QoE vs. Seek Frequency

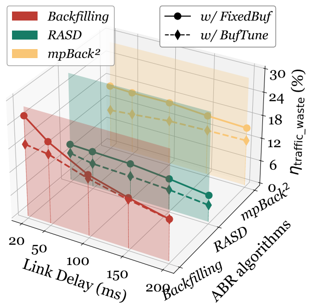


(b) Bandwidth Waste vs. Seek Frequency

Fig. 12: Effect of *BufTune* on ABR Algorithms Across Seek Frequencies.

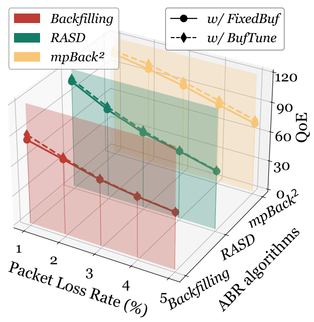


(a) QoE vs. Link Delay

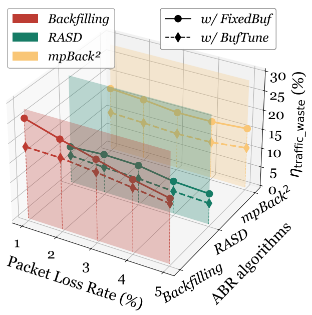


(b) Bandwidth Waste vs. Link Delay

Fig. 13: Effect of *BufTune* on ABR Algorithms Across Link Delays.



(a) QoE vs. Packet Loss Rate



(b) Bandwidth Waste vs. Packet Loss Rate

Fig. 14: Effect of *BufTune* on ABR Algorithms Across Packet Loss Rates.

*2) **Impact of Link Delay on BufTune**:* To assess how network heterogeneity affects the performance of *BufTune*, we vary the link delay of the faster path from $20ms$ to $200ms$ while keeping the slow path fixed. Other settings follow the baseline experiments with 5 user seek events per session.

We make two key observations from Fig. 13. First, when the fast path has low link delay, *BufTune* achieves a slight but consistent QoE improvement. As the link delay increases, the improvement diminishes and eventually aligns with the performance of the non-*BufTune*-enabled setup. This trend stems from the fact that buffer shrinking—triggered by seek events—enables bandwidth to be reallocated toward enhancement layers, thereby improving playback quality. Yet, as the network delay increases, the system's ability to timely retrieve enhancement layers is limited, reducing the effectiveness of this reallocation. Nonetheless, *BufTune* consistently preserves QoE even under less favorable network conditions, demonstrating its robustness across a range of heterogeneous environments. Second, traffic waste consistently decreases with increasing delay across all ABR algorithms, even without *BufTune*. This trend is expected: higher link delays naturally hinder timely enhancement-layer delivery, thereby reducing the likelihood of data being prefetched but never played. As a result, overall traffic waste diminishes. Consequently, the relative benefit of *BufTune* in alleviating wasted traffic also decreases, not because the mechanism is less effective, but because there is simply less waste left to optimize.

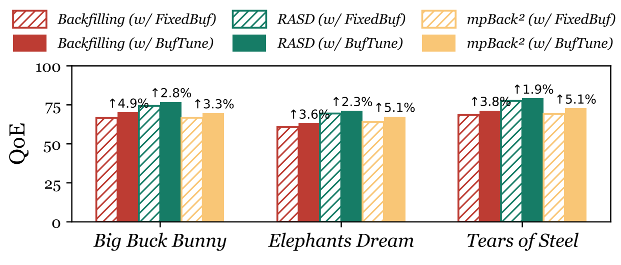
*3) **Impact of Packet Loss Rate on BufTune**:* Figure 14 presents the effect of *BufTune* on various ABR algorithms under packet loss rates ranging from 1% to 5%. This range spans from mild degradation to severely poor network conditions. As shown in Fig. 14(a), while overall QoE deteriorates across all ABR algorithms with rising loss, *BufTune* consistently maintains superior or comparable QoE compared to the non-*BufTune*-enabled setup, underscoring its effectiveness in mitigating the impact of unreliable networks. Meanwhile, as packet loss increases, overall traffic waste naturally decreases, since fewer segments are successfully delivered and buffered. This limits the scope for unnecessary prefetching. Still, *BufTune* consistently achieves lower waste across all loss rates, highlighting its efficiency even under degraded network conditions. Together, these results confirm that *BufTune* delivers both improved playback quality and better bandwidth efficiency across a range of lossy network scenarios.

*4) **Generality Across Multiple Videos**:* We further include two publicly available SVC-encoded videos, *Elephants Dreams* [63] and *Tears of Steel* [62], to evaluate *BufTune*'s generality. Results are averaged over 10 diverse network scenarios, capturing a broad range of realistic conditions. As shown in Fig. 15(a) and 15(b), *BufTune* consistently enhances QoE and traffic efficiency across all three videos and ABR algorithms. On average, it achieves up to 5.1% QoE improvement and 38.5% reduction in traffic waste, highlighting its robustness and generality across diverse content characteristics.

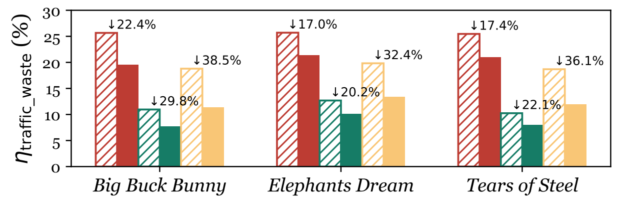
*5) **Fixed Buffer Size Impact**:* To assess whether simple buffer size reduction suffices, we conduct an extended comparison between *BufTune* and *FixedBuf* settings with varying buffer sizes (10, 15, 20, and 25 segments). Fig. 16 presents the results across three ABR algorithms. Firstly, although the traffic waste ratio does decrease with smaller buffers (e.g., from 25.4% to 23.3% for *Backfilling*), this gain is limited in magnitude. Still, it falls short of *BufTune*'s reductions (e.g., 19.7% for *Backfilling*). Secondly, overall QoE remains relatively stable across fixed buffer sizes. Nonetheless, *BufTune* consistently achieves higher QoE, especially in $mpBack^2$ (59.5 vs. 56.7) and *Backfilling* (66.9 vs. 63.8), by balancing buffer usage adaptively. These findings illustrate the limitations of static fixed-size strategies and underscore the advantages of dynamic buffer adaptation.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

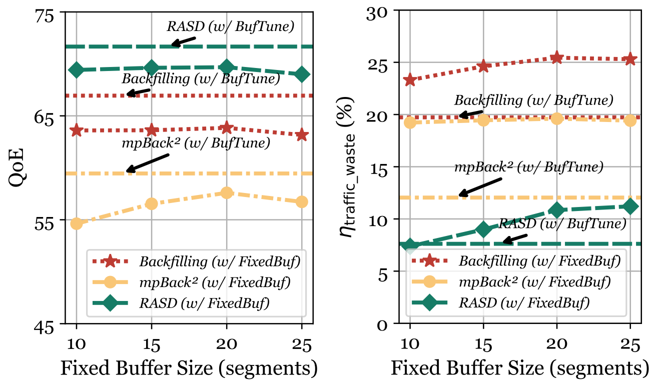


(a) QoE Performance



(b) Traffic Wastage Performance

Fig. 15: Effect of *BufTune* on ABR Algorithms Across three Videos, shown in (a) overall QoE and (b) *traffic waste ratio*. *Each group corresponds to a video; within each, six bars represent ABR algorithms with and without BufTune (hatched: w/ FixedBuf, solid: w/ BufTune).*



(a) QoE Performance (b) Traffic Wastage Performance

Fig. 16: Performance of *BufTune* versus *FixedBuf* under multiple fixed buffer sizes (10, 15, 20, and 25 segments), with results shown for (a) Overall QoE and (b) Traffic Waste Ratio. *Since BufTune dynamically adjusts the buffer size and is not bound to a fixed value, its results are plotted as horizontal reference lines corresponding to each ABR algorithm, enabling direct comparison in both QoE and Traffic Waste Ratio.*

6) Sensitivity Analysis of Hyperparameters: We evaluate the impact of four key hyperparameters in *BufTune*, including the decay aggressiveness coefficient (β), scaling coefficient for bandwidth surplus (ξ), base increment factor (δ), and sliding time window length (L). Fig. 17 presents the QoE and *traffic waste ratio* trends across different ABR algorithms.

For β, we observe that increasing its value leads to a consistent decline in QoE, as overly aggressive buffer shrinkage reduces playback stability. Meanwhile, *BufTune*'s ability to reduce traffic waste also weakens, especially when $\beta > 0.4$. For ξ, smaller values (e.g., 0.1–0.3) cause insufficient buffer growth and more frequent rebuffering under bandwidth constraints, while larger values stabilize QoE but slightly increase traffic waste due to prefetching. Similarly, increasing δ yields marginal QoE gains for most ABRs but diminishes *BufTune*'s advantage in traffic waste reduction due to overly fast buffer expansion. Finally, variations in the seek window length L show minimal impact on QoE or traffic waste for most ABR algorithms, except for a slight QoE drop in *mpBack²*, where conservative shrinkage may lead to unnecessary base-layer buffering. These trends confirm that *BufTune* remains effective across a wide range of parameter settings and that moderate configurations balance QoE and traffic efficiency.

V. Related Work

With the growing diversity of video applications, traffic waste has become a pressing issue across various streaming scenarios. Existing efforts to reduce unnecessary data transmission can be broadly categorized by video type: (i) 360° video streaming [64]–[76], (ii) short video streaming [36], [77]–[82], (iii) live streaming [74], [83]–[86], and (iv) video-on-demand (VOD) streaming [11], [13], [18], [35], [37]–[39].

A. 360° video streaming

Among these, 360° video streaming has received significant attention due to its high bandwidth demand and spatial redundancy. Prior work in this domain can be further grouped into three categories based on their underlying problem analysis.

Tile-Level Redundancy Due to Inaccurate Viewport Prediction and Inefficient Tile Scheduling. A significant body of research has addressed tile-level redundancy in 360° video streaming, particularly focusing on the unnecessary transmission of tiles outside the user's actual viewport, largely due to inaccurate viewport prediction and suboptimal tile selection [64], [65], [66], [67], [75], [76]. To tackle this, CUB360 [64] leveraged KNN-based clustering of historical viewing trajectories to enhance viewport prediction, while BAS-360° [65] employed online tile importance estimation and bandwidth-aware scheduling to deprioritize non-essential tiles. TBRA [66] applied a greedy algorithm to allocate bitrate based on the rate-distortion efficiency of tiles, aiming to enhance resource utilization. Additionally, OMMS [67] further identified that sudden viewpoint shifts during chunk playback are a major contributor to redundancy. To mitigate this, the framework incorporates both master and supplemental scheduling phases to dynamically fetch critical tiles during playback.

Tile-Level Waste Caused by Scheduling Inefficiencies and Redundant Delivery. Even with accurate viewport prediction and tile selection, system-level inefficiencies—such as suboptimal transmission scheduling, underutilization of protocol features, and lack of multi-user coordination—still cause significant traffic waste [68]–[70]. [68] analyzes tile requests under the HTTP/2 and reveals a mismatch between client-side prioritization and server-side stream scheduling. To address this, it proposes a hybrid control scheme that aligns QoE-aware priorities with HTTP/2 stream scheduling.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

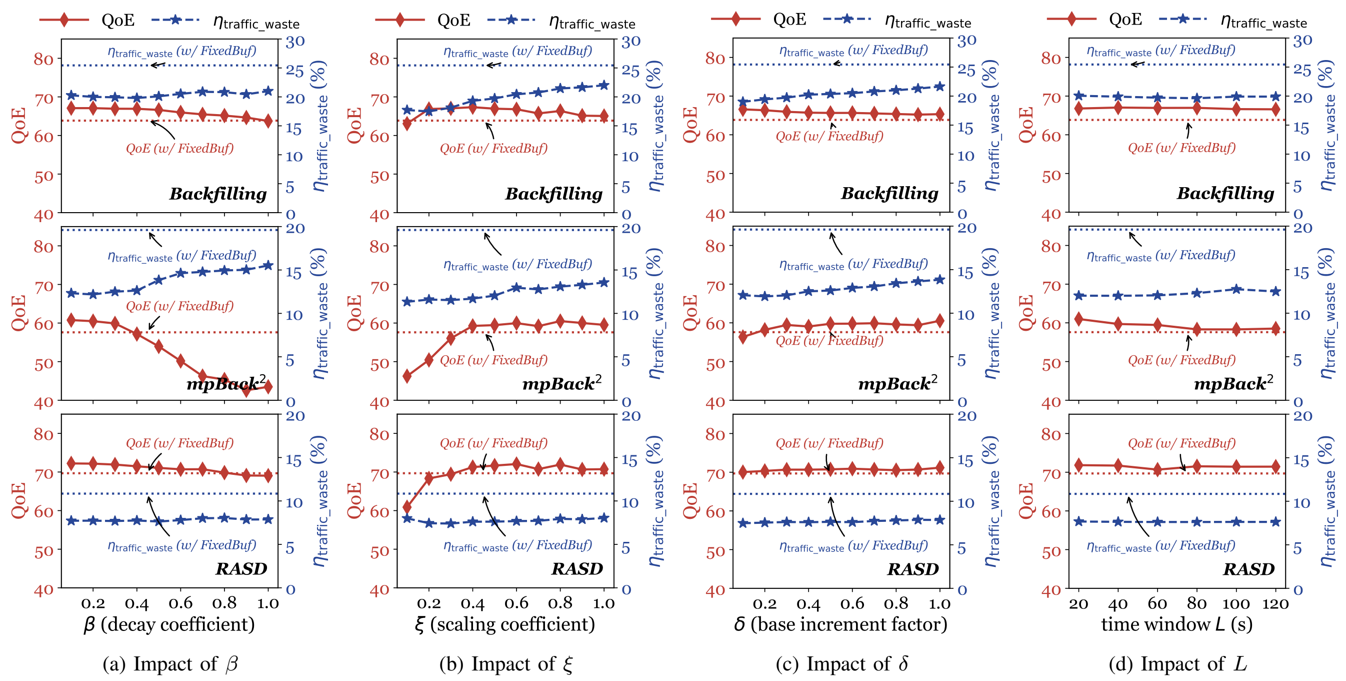


Fig. 17: Hyperparameter Sensitivity and Robustness Analysis with respect to β (decay coefficient), ξ (scaling factor), δ (ase increment factor), and L (time window).

TABLE III: Comparison of Traffic Waste Reduction Approaches Across Video Types

Category	Prior Works	Behavior Targeted	Input Basis	ABR Agnostic
360° Video Streaming	[64]–[76]	Tile/view redundancy	FoV prediction, tile priority	✗
Short Video Streaming	[36], [77]–[82]	inter-video Swiping	Watch probability, net. stats	✗
Live Streaming	[83]–[85]	Viewer-absent waste	Viewer presence, uplink/queue state	✗
Video-on-Demand	[11], [13], [35] [18], [37]–[39]	Early departure, redundancy	User behavior, QoE, content features	Partial*
BufTune (our work)	—	**In-session seeking**	**In-session interaction signals**	✓

*Some VOD works (e.g., SkipStreaming, *BufTune*) operate independently of ABR logic, while others tightly couple with it.

Building on bitrate adaptation, [69] identified the limitations of static configuration under dynamic network conditions and introduced a contextual bandit-based online selector to enable real-time bitrate policy adjustment. [70] focused on multi-user scenarios and observed that tile requests across users exhibit a high degree of overlap. To address this, DACOD360 enables tile reuse and distribution across users through a collaborative tile ownership graph and online scheduling.

Bitrate Over-Provisioning Due to the Lack of Perceptual Quality Control. Traditional QoE optimization often targets metrics like Peak Signal-to-Noise Ratio (PSNR) or Structural Similarity Index Measure (SSIM), which may increase bitrate in visually saturated regions, causing perceptual redundancy [71]. To reduce traffic without affecting user experience, some studies introduce perceptual models or reconstruction-based strategies [71], [72], [73]. [71] utilized SVC-based tile encoding with BL and ELs, loading EL data only for the current viewport, thereby maintaining perceptual quality while saving traffic. VertexShuffle [72] proposed a spherical super-resolution to enhance resolution selectively within the viewport, avoiding full-frame high-resolution transmission. In a related line of work, JUST360 [73] introduced tile-level bitrate control based on the concept of Just Noticeable Difference (JND) for the first time. A deep learning model predicts whether users can perceive quality differences, and only tiles below the JND threshold are enhanced, effectively avoiding bandwidth waste caused by quality oversupply.

While the above approaches have made significant progress in mitigating traffic waste for 360° video, they are generally based on assumptions of viewpoint continuity and spatial consistency, often relying on mechanisms such as FoV prediction and frame-by-frame tile loading. However, in VOD scenarios, especially under frequent seeking, user behavior becomes unpredictable, rendering tile scheduling ineffective and hindering the system's ability to adapt to sudden changes in playback position. More importantly, *the spatially continuous design of 360-degree video fundamentally contrasts with the segment-based access patterns of VOD*, making it challenging to apply existing optimization techniques in interactive VOD scenarios.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

B. short video streaming

To address bandwidth waste caused by the "watch-and-swipe" behavior commonly observed on short-form video platforms such as TikTok, various optimization strategies have been proposed. Based on the sources of traffic waste, existing efforts can be broadly categorized into three groups.

Redundant prefetching triggered by user swiping. This line of research attributes bandwidth waste primarily to excessive preloading caused by user swiping behavior, and thus focuses on controlling the scale and priority of preloading through user watch probability modeling [77], [78], [79]. Approaches such as APL [77], and PDAS [78] leverage predictions of user interest and retention to dynamically adjust caching strategies for videos in the recommendation queue, thereby avoiding the preloading of content unlikely to be viewed. Building on this, [79] combines reinforcement learning with multi-path download strategies to enable intelligent preloading in P2P CDN architectures.

Inefficient Download Control and Scheduling. Even with accurate user behavior prediction, inefficient download scheduling can still lead to traffic waste [80], [81], [36]. DAM [80] emphasized coordination between chunk-level downloading, pause control, bitrate adaptation, and watch probability to improve scheduling accuracy. [81] proposed a unified QoE–bandwidth metric and developed a simulator that models swipe behavior and multi-video downloading to evaluate different scheduling strategies. DUASVS [36] trained multiple adaptation models using Asynchronous Advantage Actor-Critic (A3C) [87] to dynamically select bitrates and prefetch thresholds, adjusting to varying network conditions and watch durations. Each model leverages features such as buffer level, elapsed watch time, and past bitrate decisions to jointly determine segment bitrate and prefetch thresholds.

Joint Optimization in Complex Preloading Scenarios. This line of work explores coordinated decisions across video and bitrate dimensions. A representative work, Incendio [82], decomposes the Short-form Adaptive Bitrate Recommendation (SABR) problem into two sub-tasks: video scheduling and bitrate control, which are jointly handled by a multi-agent reinforcement learning algorithm. This design significantly improves both generalization and convergence speed.

While the above methods perform well for short video streams, these approaches generally rely on user behavior models based on "watch-and-swipe" interactions, making them unsuitable for VOD scenarios characterized by arbitrary seeking. *Unlike short video users who follow a preset order, VOD users often seek within a single long video at unpredictable points and finer temporal granularity.*

C. live streaming

Some studies focus on traffic waste in live streaming [83]–[85]. For instance, Li et al. [83] investigate upstream traffic waste in mobile live streaming, where data is transmitted before viewers arrive or after they leave. They propose a viewer-triggered upload strategy that activates streaming only when viewers are present, effectively reducing redundant data without impacting latency. In contrast, TrafAda [84] targets downstream waste, modeling delivery cost via queue length, channel conditions, and bitrate. It uses Lyapunov optimization to minimize traffic cost while maximizing user-perceived quality. EDGEOPT [85] conducts large-scale measurements and reveals significant waste from viewer-absent scenarios. It proposes system-level optimizations, such as adaptive suppression of unnecessary uploads to enhance resource efficiency.

However, approaches designed for live streaming are tailored to real-time, latency-sensitive scenarios and typically rely on viewer presence as the basis for traffic optimization. *Such approaches assume a continuous, forward-only delivery flow that aligns with the presence or absence of live viewers, whereas VOD streaming enables on-demand access to pre-recorded content with arbitrary seeking across the timeline.*

D. video-on-demand (VOD) streaming

While many streaming optimizations aim to maximize QoE under the assumption of linear playback [27]–[30], [43], [45], [46], [88]–[90], an emerging line of work focuses on reducing traffic waste in VOD scenarios, where non-linear seeking such as skipping intros/outros, fast-forwarding through uninteresting segments, and early departure are common [13], [17]–[19], [91]. These efforts can be broadly categorized into two groups: *user behavior prediction* and *bandwidth-efficient ABR adaptation*.

User Behavior Prediction. Several works aim to reduce unnecessary prefetching by predicting user actions based on historical behavior [11], [13], [35]. For example, BB [11] leverages historical user behavior to divide video viewing into two distinct phases: a "browsing phase" with high early departure likelihood and a "viewing phase" with stable engagement. Based on this model, it applies conservative bitrate control during the early phase to reduce bandwidth waste, then transitions to progressive downloading as user commitment increases. SkipStreaming [13] targets episodic content and reduces bandwidth waste by identifying and skipping structurally redundant segments such as intros, recaps, and outros. It performs fine-grained cross-episode matching using audio-visual similarity to detect repeated patterns, allowing clients to selectively skip or avoid downloading redundant regions. DT [35] jointly predicts future throughput and skip behavior to guide buffer threshold selection and content preloading.

Bandwidth-Efficient ABR Adaptation. Some studies enhance ABR logic to balance quality and bandwidth efficiency [18], [37]–[39]. One line of work focuses on reducing bandwidth waste caused by early user departures [18], [37]. PSWA [18] identifies a fundamental trade-off between improving playback QoE and reducing bandwidth consumption. To navigate this trade-off, it allows content providers to define QoE preferences and acceptable degradation thresholds, enabling adaptive bitrate according to service-level objectives. BEABR [37] reduces traffic waste caused by early user departures by introducing post-download delays and leveraging reinforcement learning to guide bitrate selection. It makes decisions based on user watch progress, buffer occupancy, and real-time throughput, aiming to delay speculative downloads when departure likelihood is high while preserving overall QoE.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

While both target early departure, PSWA takes a provider-centric approach via configurable service-level trade-offs; in contrast, BEABR adopts a client-side strategy that dynamically adjusts downloads based on user watch progress.

Another direction emphasizes content-driven optimization, where bitrate adaptation is guided by intrinsic video characteristics rather than user behavior [38], [39]. TARA [39]uses model predictive control to jointly optimize rebuffering, quality switching, and traffic cost. It models QoE evolution over time and incorporates traffic cost to limit unnecessary data use. Unlike behavior-driven methods, it assumes continuous playback and does not address user-induced waste such as seeking or early exits. CAST [38] improves perceptual quality under data constraints using multi-agent reinforcement learning. It dynamically allocates bitrate based on scene intricacy, assigning higher quality to complex regions and lower to simpler ones to maximize visual quality within limited bandwidth.

While the above work has made significant progress in reducing traffic waste through user behavior prediction and ABR optimization, most approaches assume relatively structured or predictable interaction patterns such as early departure, intro skipping, or linear playback. However, real-world VOD usage frequently involves *continuous in-session seek behavior in long-form VOD playback*, where users arbitrarily jump across the timeline to revisit, skip, or fast-forward through content. These interactions are highly personalized, transient, and difficult to anticipate using offline modeling or static rules. *BufTune* addresses this overlooked challenge by operating at a distinct system layer: it adaptively adjusts buffer size in real time based solely on observed user interactions. As such, our approach is orthogonal and complementary to these methods, and not directly comparable in evaluation.

VI. Discussion and Limitations

To provide a complete understanding of our design scope, we we next discuss where *BufTune* fits within the broader ecosystem of traffic management strategies, including its limitations relative to server-side, network-level, and client-based approaches (VI-A), as well as scenarios where it may be less effective, such as live or short-video streaming (VI-B).

A. Where BufTune Fits: Strengths and Limitations

BufTune is designed to mitigate traffic waste caused by user interactions (e.g., seeks or early exits) while preserving, or even improving playback QoE, operating entirely at the client side. Approaches also exist at the server, ISP, and other client levels, and can play a role in mitigating such traffic inefficiencies from different perspectives.

Server-side mechanisms like rate shaping or pacing (e.g., capping TCP windows), aim to reduce peak load and overall bandwidth costs. However, they lack visibility into session-level user behavior, making them less effective at suppressing per-user waste. Man-in-the-middle (MITM) interventions by ISPs, such as DPI, once offered coarse traffic control based on content but are now largely obsolete due to widespread adoption of encryption (i.e., HTTPS). Moreover, such interventions often interfere with client-side ABR logic, potentially degrading QoE. Client-side approaches, including behavioral forecasting or content-aware prefetching, have direct visibility into user interactions. While these methods offer proactive waste reduction, they often rely on accurate behavioral models and may incur complexity or misprediction risks.

In contrast, *BufTune* responds directly to observed user interactions. It avoids speculative downloads without relying on prediction, offering a lightweight, deployable complement to existing solutions. While it lacks global coordination and remains reactive rather than predictive, it is effective in interactive VOD scenarios.

B. Applicability Limits

While *BufTune* is designed for VOD scenarios and targets traffic waste caused by unplayed prefetched content under interactive viewing patterns, its applicability is limited in other streaming modalities.

Live streaming. *BufTune* adapts the buffer based on intra-session behavior: when frequent seeking is observed, it shrinks the buffer to avoid downloading high-bitrate segments unlikely to be watched; when playback is stable, it gradually enlarges the buffer to maintain robustness. In live streaming, however, user-initiated seeking is typically absent, so the trigger signal for *BufTune* is missing. Furthermore, the playback buffer in live streaming is *latency-bounded* and dynamically managed to remain near the live edge (a few seconds for low-latency profiles, and tens of seconds for standard-latency settings) [92]–[94]. Consequently, aggressive buffer resizing offers limited benefit and may disrupt alignment with the live edge, increasing the risk of stalls or end-to-end latency drift.

Short video streaming. Short-video sessions comprise very short items (often on the order of tens of seconds) where *inter-item* navigation (skip/next) dominates and *intra-item* seeking is relatively rare [95]. To ensure seamless transitions, playback stacks typically prioritize prefetch of upcoming items rather than deep buffering of the current one [36]. In this regime, shrinking the current item's buffer yields limited traffic savings and may negatively affect startup latency for subsequent content. These characteristics make *BufTune* less effective in short video streaming.

VII. Conclusion and Future Work

This paper presents *BufTune*, a seek-aware buffer tuning mechanism that significantly reduces bandwidth waste in interactive VOD streaming without compromising QoE. By dynamically adjusting the playback buffer in response to real-time user seek behavior, *BufTune* effectively limits unnecessary segment prefetching while preserving, and in some cases enhancing, playback QoE. Built on a modular design comprising a *User Behavior Detector* and a *Buffer Tuner*, *BufTune* operates independently of ABR algorithms and requires no server-side modifications, making it both easily deployable and broadly applicable. Extensive experiments validate the effectiveness of *BufTune* in reducing traffic waste while maintaining high QoE across diverse ABR algorithms and network conditions.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

More broadly, *BufTune* illustrates how user behavior can be harnessed to drive intelligent, network-efficient adaptation at the edge. Its modular and ABR-agnostic design makes it well-suited not only for interactive VOD streaming but also for applications such as Extended Reality (XR) streaming, where efficient, responsive delivery remains a persistent challenge.

REFERENCES

[1] Cisco Systems, "Cisco Annual Internet Report (2018–2023)," 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/

[2] X. Li, J. Wei, H. Wang, L. Dong, R. Chen, C. Yi, J. Cai, D. Niyato, and X. Shen, "Towards Intelligent Transportation with Pedestrians and Vehicles In-the-Loop: A Surveillance Video-Assisted Federated Digital Twin Framework," *IEEE Network*, 2025.

[3] P. Zhou, Y. Xie, B. Niu, L. Pu, Z. Xu, H. Jiang, and H. Huang, "QoE-Aware 3D Video Streaming via Deep Reinforcement Learning in Software Defined Networking Enabled Mobile Edge Computing," *IEEE Trans. Network Sci. Eng.*, vol. 8, no. 1, pp. 419–433, 2021.

[4] Y. Jiang, J. Ye, L. Zhou, X. Ge, J. Kang, and D. Niyato, "Multi-Modal Stream Integrity Transmission Strategy for Multi-User Wireless Metaverse," *IEEE Trans. Commun.*, 2025.

[5] S. Yang, P. Yang, J. Chen, Q. Ye, N. Zhang, and X. Shen, "Delay-Optimized Multi-User VR Streaming via End-Edge Collaborative Neural Frame Interpolation," *IEEE Trans. Network Sci. Eng.*, vol. 11, no. 1, pp. 284–298, 2024.

[6] B. Wang, M. Su, W. Wang, K. Chen, B. Liu, F. Ren, M. Xu, J. Liu, and J. Wu, "Enhancing Low Latency Adaptive Live Streaming Through Precise Bandwidth Prediction," *IEEE/ACM Trans. Netw.*, vol. 32, no. 6, pp. 4676–4691, 2024.

[7] B. Cheng, M. Wang, X. Lin, and J. Chen, "Context-Aware Cognitive QoS Management for Networking Video Transmission," *IEEE/ACM Trans. Netw.*, vol. 29, no. 3, pp. 1422–1434, 2021.

[8] Z. Yongfei, Z. Yunsheng, Y. Chun, Q. Shiyin, and H. Zhihai, "Content-Aware Opportunistic Packet Scheduling with Dynamic Routing Algorithm for Multi-path Video Streaming over Mesh Networks," *Chin. J. Electron.*, vol. 21, no. 2, pp. 270–276, 2012.

[9] Z. Yang, J. Miao, T. Zhang, X. Tang, J. Kang, and D. Niyato, "QoE Maximization for Video Streaming in Cache-Enable Satellite-UAV-Terrestrial Network," in *Proc. ICC*, 2024.

[10] SANDVINE, "2024 Global Internet Phenomena Report," https://www.sandvine.com/phenomena/, Tech. Rep., 2024.

[11] L. Chen, Y. Zhou, and D. M. Chiu, "Smart Streaming for Online Video Services," *IEEE Trans. Multimedia*, vol. 17, no. 4, pp. 485–497, 2015.

[12] ——, "A study of User Behavior in Online VoD Services," *Comput. Commun.*, vol. 46, pp. 66–75, 2014.

[13] W. Liu, X. Yang, Z. Li, and F. Qian, "SkipStreaming: Pinpointing User-Perceived Redundancy in Correlated Web Video Streaming through the Lens of Scenes," in *Proc. MM*, 2023, pp. 3944–3953.

[14] K. Hu, H. Yang, Y. Jin, J. Liu, Y. Chen, M. Zhang, and F. Wang, "Understanding User Behavior in Volumetric Video Watching: Dataset, Analysis and Prediction," in *Proc. MM*, 2023, pp. 1108–1116.

[15] J. Carlton, A. Brown, C. Jay, and J. Keane, "Using Interaction Data to Predict Engagement with Interactive Media," in *Proc. MM*, 2021, pp. 1258–1266.

[16] Netflix, "Looking back on the origin of skip intro, five years later," https://about.netflix.com/en/news/looking-back-on-the-origin-of-skip-intro-five-years-later, 2022.

[17] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," in *Proc. ACM SIGCOMM*, 2014, pp. 187–198.

[18] G. Zhang, K. Liu, H. Hu, V. Aggarwal, and J. Y. Lee, "Post-Streaming Wastage Analysis – A Data Wastage Aware Framework in Mobile Video Streaming," *IEEE Trans. Mob. Comput.*, vol. 22, no. 1, pp. 389–401, 2021.

[19] B.-R. Chen, J. Zhu, Y. Jiang, and Y.-C. Hu, "vRetention: A User Viewing Dataset for Popular Video Streaming Services," in *Proc. ACM MMSys*, 2024, p. 313–318.

[20] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, 2007.

[21] J. M. Boyce, Y. Ye, J. Chen, and A. K. Ramasubramonian, "Overview of SHVC: Scalable Extensions of the High Efficiency Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 20–34, 2016.

[22] W. Li, J. Huang, Q. Su, W. Jiang, and J. Wang, "VASE: Enhancing Adaptive Bitrate Selection for VBR-Encoded Audio and Video Content With Deep Reinforcement Learning," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14 889–14 902, 2024.

[23] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann, "Bitrate Adaptation and Guidance With Meta Reinforcement Learning," *IEEE Trans. Mobile Comput.*, vol. 23, no. 11, pp. 10 378–10 392, 2024.

[24] F. Lyu, J. Zhang, H. Lu, H. Wu, F. Wu, Y. Zhang, and Y. Zhang, "SynthCAT: Synthesizing Cellular Association Traces with Fusion of Model-Based and Data-Driven Approaches," *Proc. IMWUT*, vol. 8, no. 4, 2024.

[25] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro, "An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming," *IEEE J. Select. Areas Commun.*, vol. 32, no. 4, pp. 693–705, 2014.

[26] M. Vandersanden, "A Holistic Approach to Understand HTTP Adaptive Streaming," in *Proc. ACM MMSys*, 2023, pp. 333–337.

[27] S. Petrangeli, N. Bouten, T. Wauters, F. D. Turck, and P. Demeester, "Towards SVC-based Adaptive Streaming in Information Centric Networks," in *Proc. ICMEW*, 2015.

[28] S. G. Ozcan, T. Kivilcim, C. Cetinkaya, and M. Sayit, "Rate Adaptation Algorithm with Backward Quality Increasing Property for SVC-DASH," in *Proc. ICCE*, 2017.

[29] S. G. Ozcan and M. Sayit, "Multipath Transmission Aware ABR Algorithm for SVC HAS," *Comput. Commun.*, pp. 20–36, March 2023.

[30] Y. Liu, B. Jiang, T. Guo, R. K. Sitaraman, D. Towsley, and X. Wang, "Grad: Learning for Overhead-aware Adaptive Video Streaming with Scalable Video Coding," in *Proc. ACM MM*, 2020, pp. 349–357.

[31] J. Ye, S. Qin, Q. Xiao, W. Jiang, X. Tang, and X. Li, "Adaptive Bitrate Algorithms via Deep Reinforcement Learning With Digital Twins Assisted Trajectory," *IEEE Trans. Network Sci. Eng.*, vol. 11, no. 4, pp. 3522–3535, 2024.

[32] S. Duan, F. Lyu, J. Cen, J. Ren, P. Yang, and Y. Zhang, "Flexible and Effective Cellular Traffic Data Synthesis with Large Language Model," in *Proc. GLOBECOM*, 2024.

[33] F. Lyu, J. Ren, N. Cheng, P. Yang, M. Li, Y. Zhang, and X. S. Shen, "LeaD: Large-Scale Edge Cache Deployment Based on Spatio-Temporal WiFi Traffic Statistics," *IEEE Trans. Mob. Comput.*, vol. 20, no. 8, pp. 2607–2623, 2021.

[34] F. Wu, S. Wang, J. Zhou, H. Pan, C. Zhou, W. Yang, F. Lyu, and Y. Zhang, "Multi-Variate Time Series Prediction of Traffic and Users for Dynamic RRH-BBU Mapping in C-RAN," *IEEE Trans. Mobile Comput.*, 2025.

[35] S. Liu, W. Wu, S. Li, T. H. Luan, and N. Zhang, "Digital Twin-Assisted Adaptive Preloading for Short Video Streaming," in *Proc. ICC*, 2024, pp. 1431–1436.

[36] G. Zhang, J. Zhang, K. Liu, J. Guo, J. Y. Lee, H. Hu, and V. Aggarwal, "DUASVS: A Mobile Data Saving Strategy in Short-form Video Streaming," *IEEE Trans. Serv. Comput*, vol. 16, no. 2, pp. 1066–1078, 2022.

[37] H. Su, S. Wang, S. Yang, T. Huang, and X. Ren, "Reducing Traffic Wastage in Video Streaming via Bandwidth-Efficient Bitrate Adaptation," *IEEE Trans. Mob. Comput.*, vol. 23, no. 11, pp. 10 361–10 377, 2024.

[38] W. Li, J. Huang, Y. Liang, Q. Su, J. Liu, W. Lyu, and J. Wang, "Optimizing Video Streaming in Dynamic Networks: An Intelligent Adaptive Bitrate Solution Considering Scene Intricacy and Data Budget," *IEEE Trans. Mob. Comput.*, vol. 23, no. 12, pp. 12 280–12 297, 2024.

[39] A. Xiao, X. Huang, S. Wu, H. Chen, and L. Ma, "Traffic-Aware Rate Adaptation for Improving Time-Varying QoE Factors in Mobile Video Streaming," *IEEE Trans. Network Sci. Eng.*, vol. 7, no. 4, pp. 2392–2405, 2020.

[40] "Netflix," https://www.netflix.com/.

[41] "YouTube," https://www.youtube.com/.

[42] YouTube, "Add automatic video chapters to your videos," https://support.google.com/youtube/answer/9884579?hl=en, 2024.

[43] Z. Zhang, T. Cao, X. Wang, H. Xiao, and J. Guan, "VC-PPQ: Privacy-Preserving Q-Learning Based Video Caching Optimization in Mobile Edge Networks," *IEEE Trans. Network Sci. Eng.*, vol. 9, no. 6, pp. 4129–4144, 2022.

[44] J. Fan, K. Zhang, Y. Zhao, and Q. Liu, "Unsupervised video object segmentation via weak user interaction and temporal modulation," *Chin. J. Electron.*, vol. 32, no. 3, pp. 507–518, 2023.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2025.3631870

[45] F. Wu, W. Yang, J. Ren *et al.*, "Adaptive Video Streaming using Dynamic NDN Multicast in WLAN," in *Proc. IEEE INFOCOM Workshops*, 2020, pp. 97–102.

[46] J. Yuan, B. Lu, M. Hao, X. Liu, L. Song, and W. Zhang, "SpaAbr: Size Prediction Assisted Adaptive Bitrate Algorithm for Scalable Video Coding Contents," in *Proc. BMSB*, 2021.

[47] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen, "LBP: Robust Rate Adaptation Algorithm for SVC Video Streaming," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1633–1645, 2018.

[48] M. Dasari, K. Kahatapitiya, S. R. Das, A. Balasubramanian, and D. Samaras, "Swift: Adaptive Video Streaming with Layered Neural Codecs," in *Proc. NSDI*, 2022, pp. 103–118.

[49] M. Nguyen, H. Amirpour, C. Timmerer, and H. Hellwagner, "Scalable High Efficiency Video Coding based HTTP Adaptive Streaming over QUIC," in *Proc. CoNEXT Workshop (EPIQ)*, 2020, p. 28–34.

[50] A. Elgabli, M. Felemban, and V. Aggarwal, "GroupCast: Preference-aware Cooperative Video Streaming with Scalable Video Coding," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1138–1150, 2019.

[51] "Mininet." [Online]. Available: https://mininet.org/

[52] "Big Buck Bunny." [Online]. Available: https://peach.blender.org/

[53] H. Mao, "Pensieve." [Online]. Available: https://github.com/hongzimao/pensieve

[54] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis, "Trickle: Rate limiting {YouTube} video streaming," in *Proc. ATC 12*, 2012, pp. 191–196.

[55] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen, "Server-based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players," in *Proc. NOSSDAV*, 2013, p. 19–24.

[56] R. Dubin, O. Hadar, A. Noam, and R. Ohayon, "Progressive Download Video Rate Traffic Shaping using TCP Window and Deep Packet Inspection," *WORLDCOMP, Nevada, USA*, 2012.

[57] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proc. ACM SIGCOMM*, 2017, pp. 183–196.

[58] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a Long Look at QUIC: an Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols," in *Proc. IMC*, 2017, pp. 290–303.

[59] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015, pp. 325–338.

[60] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath TCP," in *Proc. NSDI*, 2012, pp. 399–412.

[61] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is not Pareto-optimal: Performance Issues and a Possible Solution," in *Proc. ACM CoNEXT*, 2012.

[62] "Tears of Steel," 2024. [Online]. Available: https://mango.blender.org/

[63] "Elephants Dream," 2024. [Online]. Available: https://orange.blender.org/

[64] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Wang, "CUB360: Exploiting Cross-Users Behaviors for Viewport Prediction in 360 Video Adaptive Streaming," in *Proc. ICME)*, 2018.

[65] M. Xiao, C. Zhou, V. Swaminathan, Y. Liu, and S. Chen, "BAS-360°: Exploring Spatial and Temporal Adaptability in 360-degree Videos over HTTP/2," in *Proc. INFOCOM*, 2018, pp. 953–961.

[66] L. Zhang, Y. Suo, X. Wu, F. Wang, Y. Chen, L. Cui, J. Liu, and Z. Ming, "TBRA: Tiling and Bitrate Adaptation for Mobile 360-Degree Video Streaming," in *Proc. MM*, 2021, p. 4007–4015.

[67] R. Xu, C. Liu, M. Hu, S. Qian, Y. Zhang, and T. Lin, "OMMS: Multiple Control based Adaptive 360° Video Streaming," in *Proc. MMSys*, 2024, p. 429–434.

[68] X. Wei, M. Zhou, S. Kwong, H. Yuan, and W. Jia, "A Hybrid Control Scheme for 360-Degree Dynamic Adaptive Video Streaming Over Mobile Devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3428–3442, 2022.

[69] M. Tang and V. W. Wong, "Online Bitrate Selection for Viewport Adaptive 360-Degree Video Streaming," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2506–2517, 2022.

[70] X. Tan, S. Wang, X. Xu, Q. Zheng, J. Yang, and S. Chen, "DACOD360: Deadline-Aware Content Delivery for 360-Degree Video Streaming Over MEC Networks," *IEEE Trans. Multimedia*, vol. 26, pp. 4168–4182, 2024.

[71] A. T. Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash, "Adaptive 360-degree video streaming using layered video coding," in *proc. IEEE VR*, 2017, pp. 347–348.

[72] N. Li and Y. Liu, "Applying VertexShuffle toward 360-degree video super-resolution," in *Proc. NOSSDAV*, 2022, p. 71–77.

[73] Z. Li, Y. Wang, Y. Liu, J. Li, and P. Zhu, "JUST360: Optimizing 360-Degree Video Streaming Systems With Joint Utility," *IEEE Trans. Broadcast.*, vol. 70, no. 2, pp. 468–481, 2024.

[74] S. Kumar, A. Franklin A, J. Jin, and Y.-N. Dong, "Seer: Learning-Based 360° Video Streaming for MEC-Equipped Cellular Networks," *IEEE Trans. Network Sci. Eng.*, vol. 10, no. 6, pp. 3308–3319, 2023.

[75] J. Li, L. Han, C. Zhang, Q. Li, and Z. Liu, "Spherical Convolution Empowered Viewport Prediction in 360 Video Multicast with Limited FoV Feedback," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 19, no. 1, pp. 1–23, 2023.

[76] L. Zhao, Y. Cui, Z. Liu, Y. Zhang, and S. Yang, "Adaptive streaming of 360 videos with perfect, imperfect, and unknown FoV viewing probabilities in wireless networks," *IEEE Trans. Image Process.*, vol. 30, pp. 7744–7759, 2021.

[77] H. Zhang, Y. Ban, X. Zhang, Z. Guo, Z. Xu, S. Meng, J. Li, and Y. Wang, "APL: Adaptive Preloading of Short Video with Lyapunov Optimization," in *proc. IEEE VCIP*, 2020, pp. 13–16.

[78] C. Zhou, Y. Ban, Y. Zhao, L. Guo, and B. Yu, "PDAS: Probability-Driven Adaptive Streaming for Short Video," in *Proc. MM*, 2022, p. 7021–7025.

[79] D. Wei, J. Zhang, H. Li, Z. Xue, Y. Peng, and R. Han, "Multipath Smart Preloading Algorithms in Short Video Peer-to-Peer CDN Transmission Architecture," *IEEE Network*, vol. 38, no. 3, pp. 285–291, 2024.

[80] S.-Z. Qian, Y. Xie, Z. Pan, Y. Zhang, and T. Lin, "DAM: Deep Reinforcement Learning based Preload Algorithm with Action Masking for Short Video Streaming," in *Proc. MM*, 2022, p. 7030–7034.

[81] X. Zuo, Y. Li, M. Xu, W. T. Ooi, J. Liu, J. Jiang, X. Zhang, K. Zheng, and Y. Cui, "Bandwidth-Efficient Multi-video Prefetching for Short Video Streaming," in *Proc. MM*, p. 7084–7088.

[82] Y. Li, Q. Zheng, Z. Zhang, H. Chen, and Z. Ma, "Improving ABR Performance for Short Video Streaming Using Multi-Agent Reinforcement Learning with Expert Guidance," in *Proc. NOSSDAV*, 2023, p. 58–64.

[83] J. Li, Z. Li, Q. Wu, and G. Tyson, "On Uploading Behavior and Optimizations of a Mobile Live Streaming Service," in *Proc. IEEE INFOCOM*, 2022, pp. 1299–1308.

[84] Y. Wang, D. Zhao, C. Huang, F. Yang, T. Gao, A. Zhou, H. Zhang, H. Ma, Y. Du, and A. Chen, "TrafAda: Cost-Aware Traffic Adaptation for Maximizing Bitrates in Live Streaming," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 96–109, 2024.

[85] Z. Li, J. Li, Q. Wu, G. Tyson, and G. Xie, "A Large-Scale Measurement and Optimization of Mobile Live Streaming Services," *IEEE Trans. Mob. Comput.*, vol. 22, no. 12, pp. 7294–7309, 2023.

[86] Z. Wu, L. Lu, X. Chen, Y. Ma, S. Yang, M. Wang, L. Zhong, D. O. Wu, and C. Xu, "SRA360: Super-Resolution Enhanced Adaptive 360-Degree Video Streaming for Heterogeneous Viewers," *IEEE Trans. Network Sci. Eng.*, 2025.

[87] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Commun. Surv. Tutorials*, vol. 25, no. 1, pp. 591–624, 2023.

[88] W. Yang, F. Wu, and K. Tian, "High performance adaptive video streaming using NDN WLAN multicast," in *Proc. ICN*, 2021, p. 42–51.

[89] W. Yang, J. Cao, and F. Wu, "Adaptive Video Streaming with Scalable Video Coding using Multipath QUIC," in *Proc. IPCCC*, 2021.

[90] J. Wu, W. Yang, and L. Hui, "An MPQUIC-Based Frame-Granularity Transmission Mechanism for Adaptive Video Streaming," in *Proc. GLOBECOM*, 2023.

[91] R. Li, Q. Li, Q. Zou, D. Zhao, X. Zeng, Y. Huang, Y. Jiang, F. Lyu, G. Ormazabal, A. Singh, and H. Schulzrinne, "IoTGemini: Modeling IoT Network Behaviors for Synthetic Traffic Generation," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 13 240–13 257, 2024.

[92] R. Pantos and Others, "HTTP Live Streaming, 2nd Edition (Internet-Draft)," https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis-18, 2025.

[93] DASH Industry Forum, "Low Latency Streaming — dash.js Documentation," https://dashif.org/dash.js/pages/usage/low-latency.html, 2023.

[94] Apple Inc., "Enabling Low-Latency HTTP Live Streaming (HLS)," https://developer.apple.com/documentation/http-live-streaming/enabling-low-latency-http-live-streaming-hls, 2025.

[95] X. Zuo, Y. Li, M. Xu, W. T. Ooi, J. Liu, J. Jiang, X. Zhang, K. Zheng, and Y. Cui, "Bandwidth-Efficient Multi-video Prefetching for Short Video Streaming," in *Proc. MM*, 2022.

Authorized licensed use limited to: National Huaqiao University. Downloaded on December 22,2025 at 01:25:49 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.