Original publication: Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 2024).

ARTEMIS: Adaptive Bitrate Ladder Optimization for Live Video Streaming

Farzad Tashtarian[†] Abdelhak Bentaleb[§] Hadi Amirpour[†] Sergey Gorinsky[*]
Junchen Jiang[$] Hermann Hellwagner[†] Christian Timmerer[†]

[†]*Christian Doppler Laboratory ATHENA, Alpen-Adria Universität Klagenfurt*
[§]*Concordia University* [*]*IMDEA Networks Institute* [$]*University of Chicago*

Abstract

Live streaming of segmented videos over the Hypertext Transfer Protocol (HTTP) is increasingly popular and serves heterogeneous clients by offering each segment in multiple representations. A bitrate ladder expresses this choice as a list of bitrate-resolution pairs. Whereas existing solutions for HTTP-based live streaming use a static bitrate ladder, the fixed ladders struggle to appropriately accommodate the dynamics in the video content and network-conditioned client capabilities. This paper proposes ARTEMIS as a practical scalable alternative that dynamically configures the bitrate ladder depending on the content complexity, network conditions, and clients' statistics. ARTEMIS seamlessly integrates with the end-to-end streaming pipeline and operates transparently to video encoders and clients. We develop a cloud-based implementation of ARTEMIS and conduct extensive real-world and trace-driven experiments. The experimental comparison vs. existing prominent bitrate ladders demonstrates that live streaming with ARTEMIS outperforms all baseline solutions, reduces encoding computation by 25%, end-to-end latency by 18%, and increases the quality of experience by 11%.

1 Introduction

Live streaming is an increasingly prominent variant of video streaming. Video applications keep gaining popularity in general, with the Internet experiencing a 24% increase in video traffic during 2022 [55]. Live streaming constitutes a major contributor to this growth and feeds on support by social media platforms and streaming services such as Facebook Live, Twitch, and YouTube Live. The skyrocketing rise of live streaming in all Internet traffic from less than 1% to nearly 18% during the 2015-2022 period substantiates the importance of this streaming mode.

HTTP Adaptive Streaming (HAS) comprises an attractive option for not only Video On Demand (VOD) but also live streaming. The two leading representatives of the HAS paradigm are Apple's HTTP Live Streaming (HLS) [52] and standardized Dynamic Adaptive Streaming over HTTP (DASH) [36]. Both formats have low-latency extensions [20, 26] that accept the same Common Media Application Format (CMAF) [35] for video packaging. While WebRTC [38] enables streaming with subsecond latency and creates a promising alternative to HAS formats for interactive applications, the HAS paradigm maintains its dominance in live streaming distribution [63] due to its easy deployment, great scalability, and good performance.

In HAS, an origin server partitions a video into segments and encodes each segment into multiple representations characterized typically by a bitrate and resolution. A *bitrate ladder*, aka an *encoding ladder* or simply a *ladder*, comprises a list of the bitrate-resolution pairs. Each client, aka player, downloads from the server a *manifest* file that describes the bitrate ladder and other metadata. The video delivery is segment by segment, with the client requesting the next segment in a representation chosen by an *adaptive bitrate (ABR) algorithm* [24]. The choice strives to accommodate dynamic network conditions and balance conflicting performance objectives. For example, a higher bitrate might increase both video quality and stall duration because the client does not receive the segment in time for its playback.

A key metric of HAS performance is Quality of Experience (QoE) which captures the overall satisfaction of the user with the streaming service [56]. In this paper, we evaluate QoE using the model introduced by Comyco [33]. This QoE model measures video quality by means of Video Multimethod Assessment Fusion (VMAF) [45] and expresses QoE as a weighted sum of VMAF, stall duration, and VMAF instability over a sequence of segments. VMAF relies on machine learning to account for human perception, spatial and temporal video characteristics, and many other factors. Appendix A describes the QoE model in more detail.

HAS scales up to millions of clients due to two features. First, because the number of representations, rather than the number of clients, controls the encoding, storage, and bandwidth overhead, bitrate ladders contain relatively few representations, *e.g.*, Twitch and Apple use ladders with six and nine representations, respectively [21, 22]. When the network bandwidth available for a client aligns with a bitrate imperfectly, the discrepancy does not disrupt QoE as long as it lies within the Just-Noticeable Difference (JND) [18]. Second, HAS employs Content Delivery Networks (CDNs) which deploy edge caches around the world to serve end users with low latency [31].

In comparison to VOD, live streaming faces new challenges. The fundamental difference lies in the real-time operation of the end-to-end pipeline from the video ingestion at the camera

to the playback at the client. Specifically, VOD encoding is offline and free from stringent latency constraints. The state of the art in ladder construction goes beyond one-size-fits-all ladders [32, 42] and leverages video content and viewing context to build more effective ladders [28, 32, 37, 40, 47, 53, 54, 64, 65]. The construction of content-aware and context-aware ladders commonly relies on exhaustive search or other time-consuming methods. In contrast, live streaming is subject to tight constraints on end-to-end latency and calls for new designs. For example, live streams encode on a subsegment level, *e.g.*, all the way down to individual frames. Also, a live encoder is unable to assess video quality via VMAF because the computation of VMAF would introduce substantial latency which might even exceed the encoding latency. On the ABR side, L2A [39] and LoL$^+$ [22] represent prominent low-latency ABR algorithms that leverage the subsegment encoding structure and incorporate additional mechanisms. For example, LoL$^+$ includes a learning-based technique that proactively controls the playback speed by considering both current latency and buffer occupancy level so as to jointly handle stalls and achieve the latency target.

The live mode also creates opportunities for more effective streaming. In VOD, the origin server stores the video in all representations of the bitrate ladder for future streaming to a priori unknown clients. In live streaming, the server does not necessarily need to store the streams for future playback. Apart from making video storage less of a concern, this aspect of live streaming opens the possibility of dynamically configuring the bitrate ladder of the session to improve the encoding efficiency and QoE of current clients.

In this paper, we design ARTEMIS[1], a system that dynamically constructs effective ladders for the live encoder during the live video session without any pre-encoding step. The system objectives are to: (1) support end-to-end streaming latency of a few seconds, (2) provide heterogeneous clients with high QoE, (3) utilize network bandwidth efficiently with low storage and processing overhead, and (4) operate transparently with existing video codecs, players, and ABR algorithms.

ARTEMIS achieves its key innovation by collecting client-state information via CDN log files and leveraging this information to dynamically configure the bitrate ladder. The added mechanisms preserve the scalability of CDN-assisted HAS. In particular, whereas the number of representations encoded in real time constitutes the most critical resource in a live streaming system, the encoding of the live video into the representations that fully accommodate the capabilities of all individual clients in the session does not constitute a practical option, and ARTEMIS instead constructs bitrate ladders with a relatively small number of representations. Another important innovation of ARTEMIS is a mega-manifest file that advertises a large number of representations. ARTEMIS utilizes the mega-manifest to collect information about the bitrate preferences of the clients with a higher fidelity.

[1] *ARTEMIS* is an abbreviation of the following full name of the design: *A*daptive bit*RaTE* ladder opti*MI*zation for live video *S*treaming.

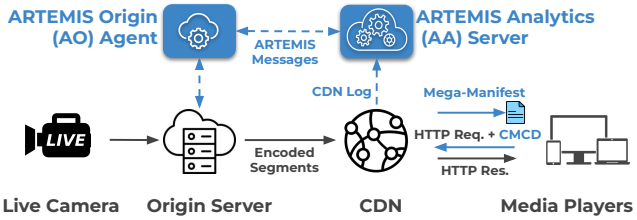

Figure 1: Conceptual architecture of live video streaming with ARTEMIS enhancements (in blue).

Figure 1 depicts the ARTEMIS architecture and its role in live streaming. ARTEMIS consists of an *ARTEMIS analytics (AA) server* and *ARTEMIS origin (AO) agent* with limited communications between these two components. Operating in real time to dynamically compose the bitrate ladder from the representations advertised in the mega-manifest, the AA server utilizes CDN logs in the Common Media Client Data (CMCD) format [7] to scalably collect client-side information about stall duration and numbers of requests for each bitrate in the mega-manifest. ARTEMIS uses the Peak Signal-to-Noise Ratio (PSNR) [34] as a metric of video quality because the latency of computing a PSNR value is negligible, within a millisecond. Specifically, based on the PSNR values computed by the live encoder for past segments, the AO agent trains a function predicting the video quality of future segments and communicates this quality indicator function to the AA server. Because the AO agent runs on the machine of the live encoder, ARTEMIS trains the quality indicator function at the AO agent, rather than at the AA server, in order to reduce traffic from the live encoder to the AA server. Utilizing the quality indicator function received from the AO agent, the AA server solves a Mixed Integer Linear Program (MILP) to update the bitrate ladder. Upon receiving the updated ladder from the AA server, the AO agent instructs the origin server to encode the subsequent segments of the video according to this new ladder. In the most likely economic realization, the streaming service provider administers both components of ARTEMIS.

We implement the ARTEMIS system in the Amazon cloud infrastructure and report an extensive real-world evaluation of ARTEMIS-enhanced live streaming. Specifically, our Python implementations of the AA server and the AO agent run in the Amazon Elastic Compute Cloud (EC2) [2]. To complete the end-to-end streaming pipeline, we utilize the Bitmovin Live Encoder [5] for the origin-server encoding of videos, Amazon CloudFront as the CDN, and multiple instances of the DASH JavaScript Player (dash.js) [27] as video clients on the CAdViSE platform [57]. The Bitmovin Live Encoder and CAdViSE also run in EC2. The live streaming experiments confirm that ARTEMIS achieves its design objectives and significantly outperforms existing

techniques for ladder construction in terms of provided QoE and network utilization while imposing low computation and communication overhead.

Our paper makes the following main contributions:

1. We design ARTEMIS, a practical system that enhances live video streaming by dynamically constructing the bitrate ladder accounting for the content complexity and network conditions. The construction leverages the clients' fine-grained bitrate preferences and stall information collected scalably through the CDN.

2. The paper reports a cloud-based implementation of ARTEMIS and an extensive real-world evaluation of its utility for end-to-end live streaming.

3. The experimental comparison vs. existing prominent bitrate ladders demonstrates that ARTEMIS delivers multi-objective improvements over the static ladders to reduce encoding computation by 25%, end-to-end latency by 18%, and increase QoE by 11%.

2 Related Work

The traditional HAS approach relies on the same bitrate ladder throughout the streaming session. The fixed ladder might be agnostic of the video content, *e.g.*, Apple's ladder [51], or depend on the content type, *e.g.*, per-title encoding by Netflix [28, 47]. One can classify recent advanced solutions into content-aware and context-aware categories.

Content-aware ladder construction accounts for the content complexity. Whereas the initial proposal of per-title encoding [28] seeks to improve the resolution for each bitrate in the ladder, [19] aims at content-aware improvements in both resolution and frame rate. [46] considers the content complexity to partition the video offline for streaming with segments of variable duration and then augments the bitrate ladder with additional representations to improve QoE via fine-grained adaptation. [40] extracts content features and applies machine learning to configure each quantization parameter (QP) based on the rate-distortion curves of different resolutions. [25] uses metrics of video quality to select a resolution for each bitrate in a perception-aware manner. [49, 50] extract low-complexity video features to set the resolutions and QPs in bitrate ladders for live streaming. While all the above solutions successfully leverage content awareness to improve bitrate ladders, [19, 28, 46] rely on brute force to construct the ladders, which is computationally untenable in live streaming. On the other hand, [25, 40, 49, 50] do not consider network conditions. In contrast, ARTEMIS accounts for both content complexity and dynamic network-conditioned client capabilities to efficiently construct dynamic ladders for live streaming.

Context-aware ladder construction addresses the importance of the context such as the network bandwidth available for the segment download by the clients. [54] models the available network bandwidth as a continuous random variable and uses its probability density function to build ladders. [43] constructs bitrate ladders by analyzing the content complexity and historical data on the available network bandwidth in order to reduce the likelihood that the user quits watching the streamed video. [60] accounts for the user population, network dynamics, video content, and other factors in its ladder construction with the objective to improve user satisfaction. [32] applies deep reinforcement learning to build ladders with features that describe the content, available network bandwidth, and storage overhead. [41] uses a Markov model to predict the bitrate of the next requested segment so as to support transcoding and ladder construction in VOD settings. [58] modifies ABR algorithms with a plug-in to obtain feedback from the clients, characterizes video quality via VMAF estimates, and evaluates the utility of the constructed bitrate ladders via trace-driven emulations. Although sharing some aspirations and techniques with ARTEMIS, the above context-aware proposals tend to make problematic design choices that jeopardize their chances for wide deployment in practice, *e.g.*, deep learning or other computationally expensive techniques unsuitable for live streaming. The key innovation of ARTEMIS as a practical scalable system for live streaming lies in collecting client-side information via CDN logs and leveraging this information to construct dynamic context-aware bitrate ladders. ARTEMIS operates transparently to ABR algorithms and uses a mega-manifest file to accurately detect the bitrate preferences of the clients.

3 Problem Description and Motivation

This paper deals with the problem of constructing improved bitrate ladders for live streaming. As shown in Figure 1, the end-to-end streaming pipeline passes through a CDN and has two ends on the origin and client sides. On the origin side, the encoder creates a bitrate ladder from the live video captured by a camera. On the client side, the ABR algorithm continuously requests segments of the video at a dynamically chosen representation, with the client subsequently buffering and playing back the received segments. In this section, we discuss the utility and challenges of using information from both origin and client sides to improve the ladder construction.

Client-side motivation. While ARTEMIS constructs the bitrate ladder based on the bitrates requested by the clients, an alternative foundation might be the network bandwidth available for the segment download. ABR algorithms typically predict the future network bandwidth through various techniques, *e.g.*, based on historical measurements of throughput and occupancy level of the playback buffer [24]. However, ABR algorithms differ greatly in their logic of selecting the bitrate for the next requested segment: whereas the greedy approach requests the highest bitrate that does not exceed the predicted available bandwidth, many ABR

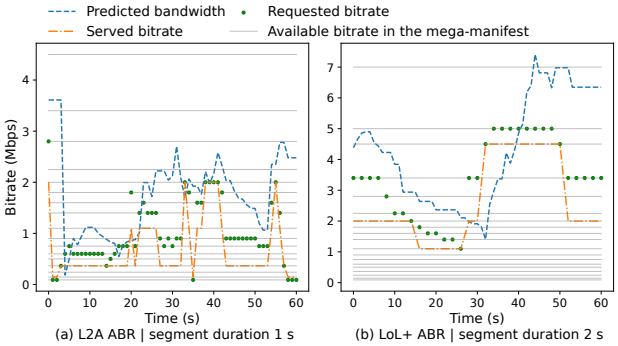

Figure 2: The bitrate ladder advertised in the mega-manifest, predicted bandwidth and requested bitrates by two ABR algorithms, and actually served bitrates at the origin side.

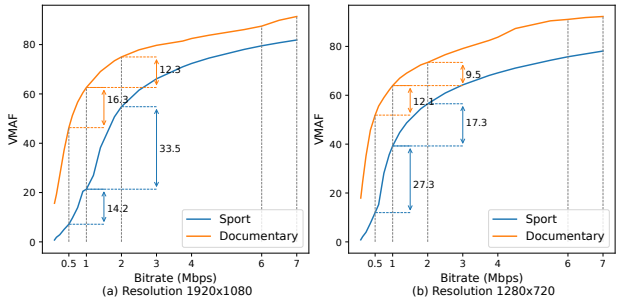

Figure 3: Dependence of video quality on the bitrate for two different content types.

algorithms choose to keep the requested bitrate stable over consecutive segments because QoE might benefit from the bitrate stability even when the bitrate is significantly below or above the available network bandwidth.

To support live streaming with low latency and utilize efficiently the limited computation, bandwidth, and storage resources, ARTEMIS constrains the number of encodings to yield fewer representations than advertised in the mega-manifest, and a client might receive a segment in a representation with a lower bitrate than requested. Although serving the segment at a lower bitrate might degrade QoE for the client, prior work adopting this technique for similar overhead reasons shows that the technique works well in conjunction with client-side ABR algorithms and supports an effective trade-off between the complexity and QoE [44, 59].

We explore the loose coupling between the client-side and origin-side logics in experiments where the mega-manifest advertises 19 representations[2]. Out of the 19 representations, the origin server actually maintains only the five representations with the bitrates of 0.145, 0.365, 1.1, 2, and 4.5 Mbps. When the client requests a representation with a different bitrate, the server delivers the maintained representation with the highest bitrate that does not exceed the requested one. The experiments evaluate the L2A ABR algorithm [39] with the segment duration of 1 s and the LTE network trace [61] as well as the LoL$^+$ ABR algorithm [22] with the segment duration of 2 s and the Cascade network trace [23]. In both experiments, the content type is animation. Figures 2a and 2b plot the results for L2A and LoL$^+$, respectively, and depict the 19 representations in the mega-manifest as horizontal gray lines, bandwidth predictions by the ABR algorithm as blue dashed lines, its requested bitrates as green dots, and actually served bitrates as orange dash-dotted lines.

[2](0.09 Mbps, 270p), (0.145 Mbps, 270p), (0.24 Mbps, 270p), (0.365 Mbps, 360p), (0.5 Mbps, 360p), (0.6 Mbps, 360p), (0.75 Mbps, 360p), (0.9 Mbps, 360p), (1.1 Mbps, 480p), (1.4 Mbps, 480p), (1.6 Mbps, 720p), (1.8 Mbps, 720p), (2 Mbps, 720p), (2.25 Mbps, 720p), (2.8 Mbps, 720p), (3.4 Mbps, 720p), (4.5 Mbps, 1080p), (5 Mbps, 1080p), and (7 Mbps, 1080p).

Figure 2 provides two relevant observations. First, the predicted available bandwidth is a poor proxy of the bitrate requested by the ABR algorithm. It is common for the discrepancy between these two values to span multiple representations. Therefore, *compared to the available network bandwidth, the bitrates requested by the clients constitute a better foundation for constructing a dynamic bitrate ladder.* Second, the L2A and LoL$^+$ algorithms adapt the bitrate effectively even though the origin server encodes segments into a smaller number of representations than advertised in the mega-manifest. Thus, *the mega-manifest empowers the origin server to detect the bitrate preference of the client with a higher accuracy and yet preserves the effectiveness of diverse ABR algorithms*. The above two observations substantiate a promise of our approach where the origin server leverages the mega-manifest technique and relies on the bitrates requested by the clients, rather than the available network bandwidth, to construct the dynamic bitrate ladder.

Origin-side motivation. Even if the origin server collects real-time fine-grained feedback about the bitrate preferences of the clients, the dynamic construction of an effective bitrate ladder requires additional information from the origin side. In particular, video quality is relevant because it strongly correlates with QoE. Figure 3 plots VMAF as a function of the bitrate for two different resolutions in the experiments that consider two content types of sport and documentary. The graphs depict the average VMAF over 100 segments, where each segment has the duration of 2 s, and show that VMAF grows sublinearly with the bitrate and greatly depends on the content type. Hence, in addition to the bitrate preferences of the clients, the construction of effective bitrate ladders should also consider the sublinear impact of the chosen bitrate on video quality as well as the specific content type. Thus, *origin-side awareness of video quality and content complexity is critical for constructing an effective bitrate ladder*.

Exploitation of content awareness in live streaming has additional challenges compared to VOD. Content-aware VOD solutions typically rely on offline search in the bitrate and resolution spaces to find a bitrate ladder where the

chosen list of bitrate-resolution pairs provides a good coverage of the VMAF space for the given content. However, the exhaustive search requires extensive computations and imposes extra latency unacceptable for real-time operation. Similarly, the computation of VMAF is untenably slow for live streaming. Therefore, *construction of a dynamic bitrate ladder for live streaming requires efficient techniques to quickly measure content complexity and select a content-aware ladder configuration.*

4 System Design

4.1 Design Principles

Based on the motivation in Section 3, we now present our ARTEMIS system and start by establishing its design principles. ARTEMIS aspires to build a dynamic bitrate ladder for a live streaming session by leveraging real-time information from both origin and client sides of the streaming pipeline. However, the provided improvements in the bitrate ladder should not undermine the traditional ability of HAS to scale up well with respect to the number of clients in the session, computation and storage overhead, and bandwidth consumption. This constraint forms a basis for our first design principle:

Design Principle 1 *Enhancements of the ladder construction should preserve scalability of live streaming and, in particular, require only affordable amounts of extra feedback from both client and origin sides.*

The existing ecosystem of live streaming is diverse in regard to deployed clients and their ABR algorithms. At the same time, ABR streaming also has standard features, e.g., the client learns about available representations by receiving from the origin server a manifest file and reacts by requesting one of the advertised representations. For ease of deployment, the proposed design should exploit common elements of end-to-end streaming and avoid modifications in the components that exist in heterogeneous instances. The above considerations lead us to the following design principle:

Design Principle 2 *ARTEMIS should seamlessly integrate with today's streaming ecosystem by leveraging its standard features and, in particular, operate transparently to heterogeneous clients and their ABR algorithms.*

Constructing an effective bitrate ladder with inexpensive overhead constitutes an important but not the only goal in live streaming. A major objective of end-to-end streaming is to support high QoE. Furthermore, because live streaming imposes tight constraints on end-to-end latency, the ladder construction should be sufficiently fast so that the streaming session fulfills the end-to-end latency constraints. Thus, we establish the final design principle for ARTEMIS:

Design Principle 3 *The ladder construction should simultaneously achieve multiple interdependent objectives that include the ladder effectiveness, reasonable overhead, high QoE, and satisfaction of end-to-end latency constraints.*

4.2 ARTEMIS Overview

We derive the ARTEMIS design from the principles established in Section 4.1. Figure 1 shows the conceptual architecture of our proposal with the AA server and AO agent as its two primary components. We envision that the provider of the streaming service administers both components and instantiates them in a cloud, *e.g.*, in EC2.

To fulfill Design Principle 1 on the client side, the AA server in ARTEMIS utilizes a CDN service to scalably collect client-state information. Akamai [1] and other major CDNs offer such services that transmit log files to third parties, *e.g.*, Conviva [6] and Datazoom [8] analytics platforms. When a streaming session uses multiple CDNs, the analytics platform obtains the client-state information separately from each individual CDN. Without loss of generality, the rest of our paper assumes that each streaming session utilizes a single CDN. In ARTEMIS, each player appends its stall information and unique player identifier (*pid*) in the CMCD format to the Uniform Resource Locator (URL) of the HTTP request message sent to the CDN edge server [3, 7]. The URL also includes the ID of the segment representation requested by the player. The CDN compiles both kinds of information provided by the players to the CDN edge servers and periodically provides the AA server with log files that contain information about stall duration and requested representations for all players in the session during the covered period. By processing the CDN log files with negligible computation overhead, the AA server extracts aggregate statistics in the form of average stall duration and request counts for each bitrate in the mega-manifest. The AA server leverages these two kinds of statistics in its computation of a new bitrate ladder.

To satisfy Design Principle 1 on the origin side, the AO agent acts as an intermediary between the origin server and the AA server in order to reduce traffic from the live encoder to the AA server. The AO agent runs on the machine of the live encoder, trains a quality indicator function on the segments produced by the encoder, and sends the quality indicator function to the AA server. The reliance of ARTEMIS on PSNR as the quality metric enables low end-to-end latency of a few seconds while accomplishing the QoE, overhead, and other objectives in compliance with Design Principle 3. Because the computation of VMAF is unacceptably slow for live streaming, Figure 14 in Appendix B shows a strong correlation between PSNR and VMAF across different segment durations and content types.

The AA server utilizes the collected origin-side and client-side information to solve an optimization program that

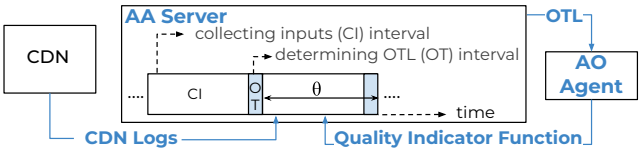

Figure 4: Time-slotted operation by ARTEMIS.

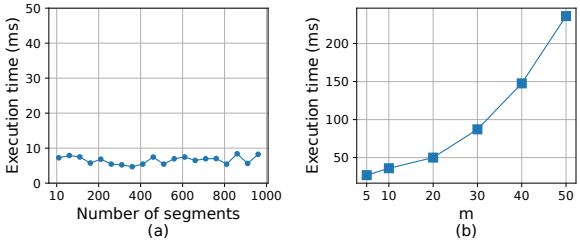

Figure 5: Execution time to: (a) train quality indicator function F for different numbers of segments and (b) solve the MILP as a function of m, the number of representations in the mega-manifest.

produces an *optimized temporary ladder (OTL)*. Because the number of representations encoded in real time constitutes the critical resource in a live streaming system, the AA server dynamically composes the OTL from only some of the representations listed in the mega-manifest and instructs the origin server through the AO agent to use the OTL as the current ladder for segment encoding. The origin server does not encode any mega-manifest representations that do not appear in the OTL. When a client requests a representation absent from the OTL, ARTEMIS instructs the origin server to stream the segment in the closest lower representation in the OTL. Hence, ARTEMIS operates transparently to the clients and origin server in accordance with Design Principle 2.

ARTEMIS relies on time slots in its internal and external communications. As shown in Figure 4, each time slot is θ seconds long and consists of two distinct intervals: the collecting inputs (CI) interval and determining OTL (OT) interval. During the CI interval, the AA server collects inputs from the CDN and AO agent. In the OT interval, the AA server uses the collected data to update the OTL. Below, we detail the methodology and explain our algorithm for determining the OTL based on the data collected during the CI interval.

4.3 OTL Selection

During each OT interval, the AA server decides whether to update the OTL. In the following, we formulate a Mixed Integer Linear Program (MILP) that computes a new OTL. Table 2 in Appendix C summarizes our notation. Without loss of generality, we assume only one fixed resolution for each bitrate in the mega-manifest. The optimal resolution for a bitrate is determinable in an online manner [50]. Set $B = \{1,2,\ldots,m\}$ indexes the m mega-manifest representations in increasing order of their bitrates, and b_i refers to the bitrate of representation i. List $R = \{r_1, r_2, \ldots, r_m\}$ consists of m nonnegative integers, where r_i denotes the number of requests for bitate b_i.

The OTL contains at most ℓ of the m representations. For each representation i, we define binary variable x_i that indicates whether bitrate b_i appears in the OTL ($x_i = 1$) or not ($x_i = 0$). When the OTL does not include bitrate b_i, and the AA server observes requests for this bitrate (*i.e.*, $r_i > 0$), ARTEMIS selects a lower bitrate in the OTL to serve the requests. To model such situations, our MILP incorporates a list of $i-1$ binary variables $Y_i = \{y_{1,i}, y_{2,i}, \ldots, y_{i-1,i}\}$ and the following two constraints:

$$\sum_{j \in B \,\&\, j<i} y_{j,i} + x_i = 1 \quad \forall i \in B, \tag{1}$$

$$\sum_{i \in B \,\&\, j<i} y_{j,i} \leq x_j \times m \quad \forall j \in B \tag{2}$$

where $y_{j,i} = 1$ indicates that ARTEMIS instructs the origin server to transmit the segment at bitrate b_j which is lower (*i.e.*, $j < i$) than requested bitrate b_i. Constraint 1 guarantees that ARTEMIS accommodates the requests for bitrate b_i by either including this bitrate into the OTL ($x_i = 1$) or serving them at a lower bitrate in the OTL ($\sum_{j \in B \,\&\, j<i} y_{j,i} = 1$). Constraint 2 enforces $x_j = 1$ when the OTL includes bitrate b_j to accommodate requests for higher bitrates than b_j.

The following constraint ensures that the OTL comprises at most ℓ representations:

$$\sum_{i \in B} x_i \leq \ell. \tag{3}$$

If all bitrates in the OTL would change every OT interval, excessive bitrate switching might degrade the QoE. To prevent frequent bitrate changes, the MILP imposes an upper limit on the number of changes in the OTL over two consecutive OT intervals:

$$\sum_{i \in B} |x_i - \bar{x}_i| \leq \beta, \tag{4}$$

where $\beta > 0$ and $\bar{x}_i \in \{0,1\}$ indicates whether bitrate b_i appears in the OTL of the previous OT interval.

The live encoder calculates the PSNR value of a segment on the fly during the encoding process, *e.g.*, by appending `-psnr` to the FFmpeg command line [10]. Based on the PSNR values collected from the encoder for previous segments, the AO agent computes a function estimating the quality of upcoming segments. Specifically, the AO agent uses linear regression to train quality indicator function F that maps a bitrate to PSNR. Figure 5a plots the execution time to train function F. The AA server leverages quality indicator function F to estimate the PSNR values for all bitrates in the mega-manifest for subsequent segments. When the OTL relies on bitrate b_j to accommodate the requests for higher bitrate b_i, the AA server utilizes function F to measure the respective change in PSNR. The following inequality introduces the nonpositive real variable q to express the quality improvement that the requested bitrates would provide compared to the bitrates chosen in the OTL:

$$q \times \sum_{i \in B} r_i \leq \sum_{i \in B} \sum_{j \in B \,\&\, j<i} r_i \times y_{j,i} \times \big(F(b_j) - F(b_i)\big). \quad (5)$$

The final constraint of our MILP brings in the nonnegative real variable s to represent the traffic reduction that happens because clients receive segments in lower representations than the requested ones:

$$s \times \sum_{i \in B} r_i \leq \sum_{i \in B} \sum_{j \in B \,\&\, j<i} r_i \times y_{j,i} \times (b_i - b_j). \quad (6)$$

For the sake of normalization, we introduce Q and S as bounds on the possible values of quality improvement q and traffic reduction s, respectively, and define the following Multi-Objective Optimization (MOO) function:

$$\text{MOO} = \alpha \times \frac{q}{Q} + (1-\alpha) \times \frac{s}{S}. \quad (7)$$

Our MILP strives to maximize the MOO function where weight α controls the relative priorities of the quality improvement vs. traffic reduction. The two individual objectives are in conflict: increasing q decreases s and vice versa. Note that $\alpha = 1$ emphasizes increasing the video quality, which conforms to the OTL selecting its bitrates as close as possible to the requested bitrates. On the other hand, $\alpha = 0$ places the only focus on minimizing the traffic, which corresponds to serving all requests with the lowest bitrate. We detail our algorithm for determining the value of α in Section 4.4. To summarize, ARTEMIS relies on the following MILP:

Maximize: MOO in Equation 7 (8)

subject to: Constraints 1 through 6 (9)

variables: $x_i,\ y_{j,i} \in \{0,1\},\ q \leq 0,$ and $s \geq 0.$ (10)

The computational complexity of the formulated MILP depends on the number of representations in the mega-manifest, and not on the number of clients in the session. The total number of variables equals $m + \frac{m(m-1)}{2} + 2$ because each bitrate b_i contributes $i-1$ binary variables $y_{j,i}$ and one binary variable x_i, with two real variables q and s completing the total. The number of individual constraints behind Constraints 1 through 6 amounts to $2m+4$. Despite being an NP-complete problem [30], the MILP is applicable for optimizing the OTL in live streaming due to the relatively low numbers of variables and constraints. We solve the MILP by using the PuLP library [14]. Figure 5b reports the respective execution time as a function of m, the number of representations in the mega-manifest.

4.4 Algorithm Details

Input processing. Algorithm 1 constitutes the core algorithm of ARTEMIS. Lines 3-6 of the pseudocode describe the operation during the CI interval when the AA server retrieves the latest version of quality indicator function F and client-state log files from the AO agent and CDN, respectively. After extracting the request counts and stall information from the CDN logs, the AA server stores the data as list $R = \{r_1, r_2, \ldots, r_m\}$ and set T that consists of tuples (pid, t_s, t_e), where pid denotes the unique player identifier included in the URLs of HTTP requests according to the CMCD specification, and t_s and t_e represent the start and end times of a stall event. A player that experiences a stall event sends the respective (pid, t_s, t_e) tuple to the CDN and requests to buffer all encoded segments that reach the CDN edge server. The time-slot duration divided by the segment duration imposes the upper limit on the number of requests that the player contributes to the R list during the time slot. For each of the players that request segments and do not report any stalls, set T includes tuple $(pid, 0, 0)$.

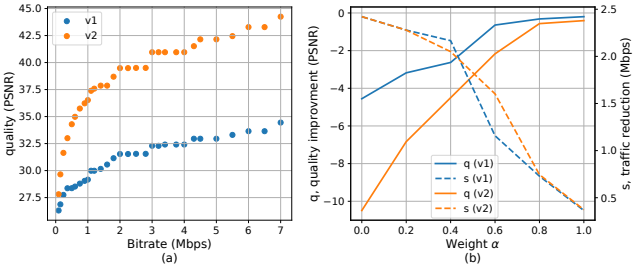

Figure 6: (a) Rate-distortion curves $v1$ and $v2$ with 30 bitrates between 0.09 Mbps and 7 Mbps and (b) impact of the α value on quality improvement and traffic reduction.

Algorithm for determining α. Before answering the question how to determine the best α value, we use Figure 6 to illustrate the impact of α on quality improvement q and traffic reduction s, *i.e.*, the two individual objectives of our MILP. Figure 6a depicts rate-distortion curves $v1$ and $v2$ associated with two different content types and $m = 30$ bitrates. For each bitrate b_i, we randomly pick a value between 50 and 100 for request count r_i and configure $\bar{x}_i$ to 0. The example also assumes $\ell = \beta = 8$. Figure 6b reveals the opposite impact of changes in α on q and s. When α equals 0, the MILP produces OTLs that offer the worst video quality, with q being about -4.2 and -10.2 for curves $v1$ and $v2$, respectively. However, the traffic reduction in this α setting is the largest, with s reaching around 2.4 Mbps for both curves. Increasing the α value improves quality and increases traffic. When α becomes 1, the MILP-produced OTLs support video quality within 0.2% and 3% of the PSNR values with the requested bitrates, and traffic reduction s drops to 0.35 Mbps for both curves $v1$ and $v2$.

Because Figure 6b shows that α significantly affects both video quality and traffic volume, ARTEMIS adjusts the α value based on the origin-side and client-side information. For instance, larger traffic with a greater α value might cause not only improved video quality but also increased stall duration, which the AA server detects by observing the increased average stall duration in the client-side feedback provided via the CDN. In adjusting the α value based on the stall information, ARTEMIS avoids dramatic changes over consecutive OT intervals so that the resulting updates in the

Algorithm 1 ARTEMIS Algorithm

1: **for** each time slot **do**
2: $R \leftarrow []$, $T \leftarrow []$, $F \leftarrow \varnothing$, $O^* \leftarrow \varnothing$
3: **while** in CI **do** ▷ CI interval starts
4: $T, R \leftarrow$ ProcessCDNlogs()
5: $F \leftarrow$ QualityFunction()
6: **end while** ▷ OT interval starts
7: $\alpha, f_1 \leftarrow$StallAnalysis(T) ▷ Algorithm 2
8: $O, q \leftarrow$ Optimization(α, O^*, R, F)
9: **if** f_1 **then**
10: SendOTLtoAOagent(O)
11: $O^* \leftarrow O$
12: **else**
13: $f_2 \leftarrow$QualityAnalysis(q, F) ▷ Alg. 3
14: **if** f_2 **then**
15: SendOTLtoAOagent(O)
16: $O^* \leftarrow O$
17: **end if**
18: **end if**
19: **end for**

Algorithm 2 StallAnalysis Function

1: **Inputs:** LastStall, StallAlpha
2: **function** STALLANALYSIS(T)
3: $l^* \leftarrow$ mean(T)
4: $\alpha \leftarrow$SelectAlpha(StallAlpha,l^*)
5: $l \leftarrow$LastStall
6: **if** $l == 0$ **then**
7: $t \leftarrow min(1, l^*)$
8: **else**
9: $t \leftarrow min(1, \frac{l^*-l}{l})$
10: **end if**
11: LastStall$\leftarrow l^*$
12: $p \leftarrow$GenerateRandom(uniform[0,1))
13: **if** $p \leq t$ **then**
14: $f \leftarrow$ **True**
15: **else**
16: $f \leftarrow$ **False**
17: **end if**
18: **return** α, f
19: **end function**

Algorithm 3 QualityAnalysis Function

1: **Inputs:** O^*
2: **function** QUALITYANALYSIS(q, F)
3: $d \leftarrow []$
4: **for** $i \in B$ **do**
5: d.append(DiffQuality(b_i,r_i,a_i,F,i))
6: **end for**
7: $q^* \leftarrow mean(d)$
8: **if** $q == 0$ **then**
9: $t \leftarrow min(1, q^*)$
10: **else**
11: $t \leftarrow min(1, \frac{q^*-q}{q})$
12: **end if**
13: $p \leftarrow$GenerateRandom(uniform[0,1))
14: **if** $p \leq t$ **then**
15: **return True**
16: **else**
17: **return False**
18: **end if**
19: **end function**

OTL do not degrade QoE for the clients, *e.g.*, due to frequent bitrate switching.

Algorithm 2 presents pseudocode of the *StallAnalysis*() function that determines the α value. The function seeks an appropriate balance between high video quality and low stall duration. In addition to set T, the function takes *LastStall* and *StallAlpha* as two other local inputs. *LastStall* refers to the average stall duration calculated during the previous time slot. *StallAlpha* is a dictionary that specifies the α value for each range of stall duration. For example, *StallAlpha*=`{(0,2):1.0,(2,'inf'):0.8}` means $\alpha = 1.0$ for the average stall duration between 0 and 2 s and $\alpha = 0.8$ for the average stall duration exceeding 2 s. Supplied by the streaming service provider, the *StallAlpha* dictionary is updatable during the streaming session.

Line 3 of Algorithm 2 computes l^* as the average stall duration in set T. Line 4 configures α according to the *StallAlpha* dictionary. With variable l initialized to *LastStall*, Lines 6-10 adjust threshold t according to the difference between l^* and l. Line 12 draws random number p uniformly between 0 and 1. If p is at most threshold t, *i.e.*, the stalling is significantly higher than in the previous time slot, Line 14 sets flag f to *True*, indicating the need for a new OTL to decrease the stalling. Otherwise, Line 16 sets flag f to *False*. Line 18 returns the α and f values. The core algorithm of ARTEMIS obtains these values as α and f_1, respectively, by calling the *StallAnalysis*() function in Line 7 of Algorithm 1.

OTL computation. Based on the determined α value, previous OTL O^*, list R of request counts, and quality indicator function F, Line 8 of Algorithm 1 computes new OTL O as described in Section 4.3. Solving the MILP also returns quality improvement q that satisfies Constraint 5.

Communication of the OTL to the AO agent. If flag f_1 is set to *True*, Line 10 of Algorithm 1 sends the new OTL from the AA server to the AO agent. Otherwise, with flag f_1 set to *False* and indicating insignificant stalling, Line 13 calls a *QualityAnalysis*() function to explore whether quality improvement justifies adopting the new OTL.

Algorithm 3 shows the pseudocode of the *QualityAnalysis*() function, with previous OTL O^* as an extra input. Lines 4-6 of Algorithm 3 measure how much the requested bitrates would improve video quality compared to the bitrates in the previous OTL. While r_i refers to the number of requests for bitrate b_i, input a_i in Line 5 indicates how many requests the previous OTL accommodates with bitrate b_i. Lines 7-12 introduce threshold t that captures the difference between q^* and q, which refer to the quality improvement for the previous and new OTL, respectively. Line 14 compares the threshold with random number p drawn uniformly between 0 and 1. The *QualityAnalysis*() function returns a *True* flag if p is at most t. Otherwise, Algorithm 3 returns a *False* flag.

When Line 13 of Algorithm 1 receives a *True* flag from the *QualityAnalysis*() function, Line 15 of the core algorithm sends the new OTL to the AO agent. With this flag f_2 set to *False*, ARTEMIS continues operation with the previous OTL. Hence, in deciding whether to update the OTL, ARTEMIS considers not only the *StallAlpha* dictionary but also video quality and stalling information across two consecutive time slots.

5 Performance Evaluation

This section describes our practical implementation of ARTEMIS as well as the evaluation methodology and results of our real-world experiments. The evaluation intends to answer the following questions: (*i*) What is the impact of ARTEMIS on QoE? How do the constructed dynamic bitrate ladders fare against existing fixed-length and content-based ladders? (*ii*) What is the influence of ARTEMIS on resource

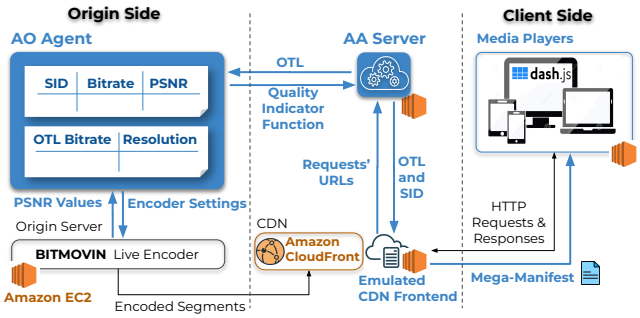

Figure 7: Cloud-based implementation of ARTEMIS.

consumption? (*iii*) How do different real-world network traces affect the ARTEMIS performance? (*iv*) How sensitive is the ARTEMIS performance to changes in the number of players, segment duration, content type, end-to-end latency, and α value?

5.1 ARTEMIS Implementation

Our real-world experimentation combines existing infrastructure with our own cloud-based implementations of the ARTEMIS functionalities. The existing infrastructure includes Amazon CloudFront as the CDN, Bitmovin Live Encoder [5] as the origin server, and CAdViSE [57] for hosting media players. Both Bitmovin Live Encoder and CAdViSE operate as EC2 implementations. We implement the ARTEMIS functionalities in EC2 as well. Figure 7 depicts our implementations and their interactions with the existing infrastructure.

Media players. CAdViSE is a cloud-based platform for automated testing of media players. We use CAdViSE to run multiple players on different kinds of EC2 machines. Each EC2 instance fetches from Docker Hub [9] a Docker container holding the DASH JavaScript Player [27]. We modify the player to send the *pid* and stall information in the CMCD format as a query argument in the URL of every HTTP request for a segment.

Emulated CDN frontend. To support the mega-manifest feature of ARTEMIS, we emulate a CDN frontend by implementing it in Python and running the frontend on an EC2 machine. The emulated CDN frontend faces the players and receives their HTTP requests. Also, the AA server provides the CDN frontend with an OTL and Segment ID (SID) to indicate the validity of the given OTL for segments with numerical IDs that are equal or greater than this SID. Before forwarding each HTTP request for a segment to the CDN, the CDN frontend checks whether the requested representation appears in the OTL associated with this segment. If the representation ID is not in the OTL, the CDN frontend changes the representation ID in the HTTP request to the ID of the representation with the closest lower bitrate in the OTL. The emulated CDN frontend also provides the AA server with CDN logs in the form of requests' URLs.

AA server. We implement the AA server in Python and, specifically, employ the Pulp library [14] to solve the MILP. Apart from determining the OTL and communicating it to the CDN frontend and AO agent, the AA server also coordinates the time-slotted operation of ARTEMIS. The two main threads of the AA server utilize Transmission Control Protocol (TCP) sockets to receive the requests' URLs and quality indicator function F from the CDN frontend and AO agent, respectively.

AO agent. Our Python implementation of the AO agent runs in EC2 on the same machine with the Bitmovin Live Encoder. The encoder uses DASH packaging, computes PSNR values during the encoding process, and records a (SID, bitrate, PSNR) tuple for each encoded segment. The computational burden of extracting the PSNR values during the encoding process is negligible [62]. The main computational task of the AO agent is to train quality indicator function F by means of linear regression. Figure 5a shows that the execution time to train function F is low, below 20 ms, and largely independent from the number of segments. In addition, based on the OTL received from the AA server, the AO agent updates the encoder settings of the Bitmovin Live Encoder. The AO agent also stores the bitrates of the previous OTL along with their resolutions. Our experiments employ predetermined resolutions for all bitrates.

5.2 Evaluation Methodology

Bitrate ladders. We consider two groups of baselines for evaluating the dynamic bitrate ladders constructed by ARTEMIS. The first group consists of common fixed-length ladders used by conventional HAS solutions for live streaming that do not account for the content type or network conditions. Table 1 presents the Theo [16], Bitmovin [4], Mux [11], Pensieve [48], and Twitch [17] ladders that comprise this first group. The second group consists of three bitrate ladders designed for particular content types and/or specific average available bandwidth. Table 3 in Appendix D describes these three ladders. Designed offline for VOD scenarios, the ILP ladder [60] targets animation videos with the average available bandwidth of the clients being around 3 Mbps. The other two baselines in the second group are the Netflix ladders for animation [12] and movie [13] as the content type and do not account for the available network bandwidth.

Network traces. Our experiments use the LTE [61], AmazonFCC [15], Cascade-5, and Cascade-20 traces. Figure 15 in Appendix D depicts the four traces for 500 s, *i.e.*, the total duration of each streaming session. Cascade-5 and Cascade-20 are synthetic network traces generated using five distinct bandwidth values of 0.5, 1, 2, 4, and 7 Mbps, where labels 5 and 20 indicate the the available bandwidth remains constant for 5 and 20 s, respectively. To allocate a network trace to each player, we convert a longer network trace into circular arrays. Then, we generate a random number uniformly distributed

Table 1: Representations advertised by the mega-manifest of ARTEMIS vs. the five static bitrate ladders.

BL \ Res.@Mbps	240p@0.145	240p@0.240	360p@0.365	360p@0.5	360p@0.6	360p@.75	360p@0.9	540p@1.0	480p@1.1	480p@1.2	480p@1.4	720p@1.6	720p@1.8	720p@2.0	720p@2.25	720p@2.5	1080p@2.8	720p@3.0	720p@3.2	720p@3.4	720p@3.75	1080p@4.0	1080p@4.3	1080p@4.5	1080p@5.0	1080p@5.5	1080p@6.0	1080p@6.5	1080p@7.0
Theo			✓					✓								✓						✓							
Bitmovin	✓		✓						✓					✓										✓					
Mux						✓		✓								✓								✓					
Pensieve			✓			✓				✓			✓				✓						✓						
Twitch				✓					✓			✓							✓		✓								✓
ARTEMIS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

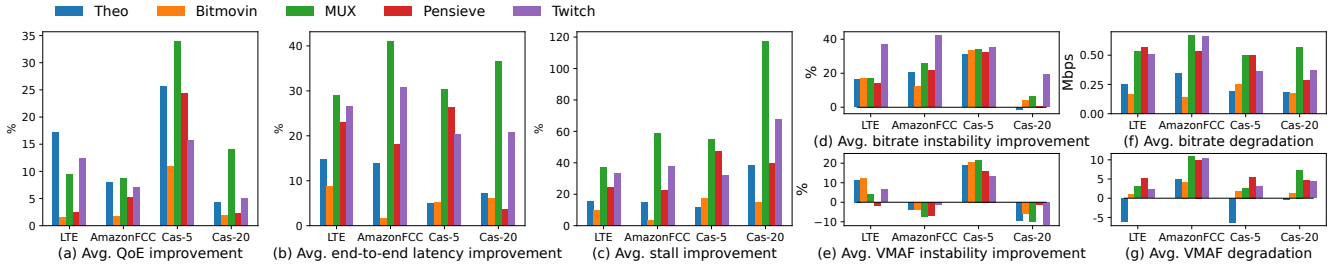

Figure 8: QoE, latency, stall, bitrate, and VMAF performance of ARTEMIS' dynamic ladders vs. the five static ladders.

between 0 and 500. This random number determines the time at which the allocated network trace starts for the specific player. Thereby, we have a distinct network trace for each player.

Content types and encoding parameters. Animation, sport, movie, and documentary are four distinct content types in our experiments. The Bitmovin Live Encoder encodes segments at a constant bitrate with segment duration of 1, 2, and 4 s and extracts the PSNR values of the segments during the encoding process.

Default ARTEMIS parameters. By default, we set maximum OTL length ℓ to 5 representations, dictionary *StallAlpha* to $\{1:[0,1], 0.9:[1,2], 0.8:[2,3], 0.7:[3,4], 0.6:[4,5], 0.5:[5,100]\}$, time-slot duration θ to 10 s, segment duration to 2 s, and target end-to-end latency to 4 s. The default number of players, ABR algorithm, and content type are 50 players, L2A, and animation, respectively.

Experimental scenarios. We experiment in two scenarios. Using the default parameter settings, Scenarios I and II evaluate ARTEMIS' dynamic ladders against the bitrate ladders from the first and second baseline groups, respectively. We also analyze performance sensitivity to changes in the experimental settings, including the ABR algorithm, content type, ARTEMIS parameters α, ℓ, and θ, and the number of players. While individual experiments in our preliminary investigation substantiate the potential of ARTEMIS in sessions with as many as 15,000 players, the evaluation in this paper accomplishes its multifaceted agenda in the settings where each session serves 10, 20, 50, or 100 players. For each experiment, we average the results over five runs.

Evaluation metrics. We evaluate QoE via the QoE model of Appendix A and also zoom in on instability of the

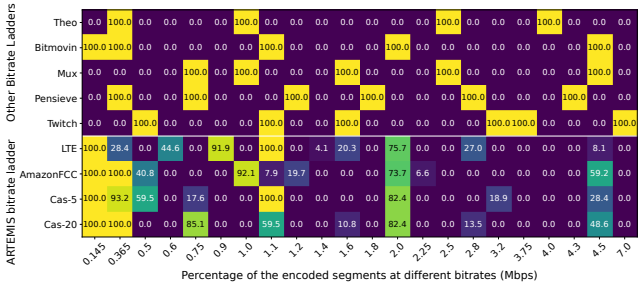

Figure 9: Bitrates used by the five static ladders and ARTEMIS' dynamic ladders on different network traces.

bitrate and VMAF. For each of these two underlying metrics, we calculate its instability as the average of the absolute differences between their consecutive values throughout a sequence of segments. Besides, we calculate ladder efficiency and resource cost. To compute ladder efficiency, we divide the average served bitrate of the clients by the average encoded bitrate of the ladder. Instead of directly measuring computation and bandwidth costs, we report encoding time and volume of traffic from the live encoder to the CDN and from the CDN to the player. We choose these metrics because the computation and bandwidth costs are proportional to the processing time and traffic volume.

5.3 Results and Analysis

Scenario I. Our evaluation of ARTEMIS starts by comparing its dynamic bitrate ladders with the five fixed-length ladders. Figures 8a-8e and, in more detail, Table 4 in Appendix E show the average relative improvement in QoE, end-to-end latency,

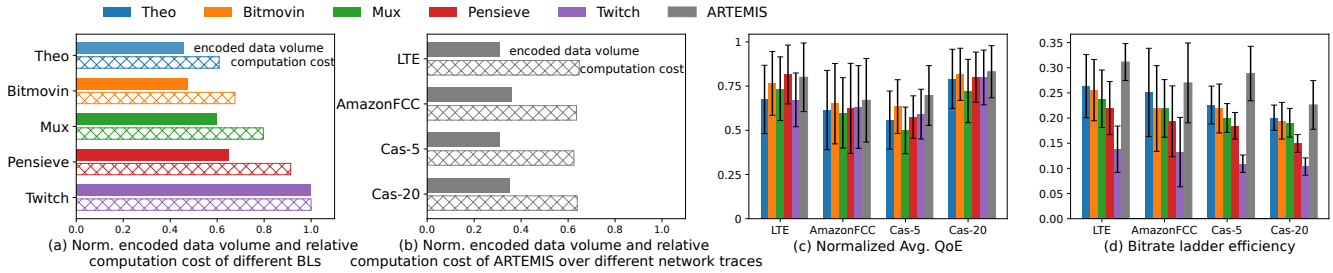

Figure 10: Computation cost, traffic volume, and ladder efficiency with ARTEMIS vs. the five static ladders.

stall duration, bitrate and VMAF instability for ARTEMIS vs. the baselines. The results demonstrate that ARTEMIS consistently improves QoE compared to the baselines, with the average QoE improvement ranging from 4% to 16.5% with respect to the Bitmovin and Mux ladders, respectively. Figure 8a shows that ARTEMIS yields the largest average QoE improvement in comparison to the Mux ladder because ARTEMIS mitigates stalls for the players by selecting lower bitrates than 0.75 Mbps, which is the lowest bitrate in the Mux ladder. Although the Bitmovin ladder includes the lowest bitrate of 0.145 Mbps, ARTEMIS outperforms this ladder as well, *e.g.*, by reducing the stall duration by 17.3% and improving QoE by 10.9% on the Cascade-5 network trace. Similarly, Figure 8b reveals that ARTEMIS outperforms the baselines by reducing the end-to-end latency. The latency reduction occurs due to shorter encoding time arising from the more effective ladder selection in ARTEMIS.

Figures 8d and 8e demonstrate that ARTEMIS decreases the bitrate instability compared to the baseline ladders, particularly on the LTE, AmazonFCC, and Cascade-5 network traces. While the VMAF instability gets worse with ARTEMIS on the AmazonFCC and Cascade-20 traces, ARTEMIS still improves the overall QoE on these traces. Furthermore, the observed increase in the VMAF instability lies within the JND, meaning that humans do not perceive the quality decrease.

Figures 8f and 8g offer further insights into the impact of ARTEMIS on the decrease in the average bitrate and VMAF compared to the baseline ladders. Although ARTEMIS serves the players with lower bitrates, the decrease in VMAF is negligible for some combinations of ladders and network traces, and ARTEMIS improves the VMAF value in other cases. ARTEMIS attains this performance by using PSNR as a proxy of VMAF in predicting the video quality of the subsequent segments. Figure 8g shows that ARTEMIS provides higher VMAF compared to the Theo ladder on the LTE and Cascade-20 network traces. The Theo ladder yields the lower VMAF values due to employing only four representations with the largest bitrate of 4 Mbps only, whereas ARTEMIS includes higher bitrates into its dynamic ladders and thereby improves the video quality.

Figure 9 reports on the bitrates used by ARTEMIS' dynamic ladders vs. the static baselines. While the static ladders always employ the same bitrates regardless of the network trace, ARTEMIS adapts its ladders to network conditions. Only the lowest mega-manifest bitrate of 0.145 Mbps always appears in ARTEMIS' OTL. The second lowest bitrate of 0.365 Mbps is always present in the OTL for the AmazonFCC and Cascade-20 network traces only. The bitrate of 1.1 Mbps is consistently in demand on the LTE and Cascade-5 traces. For all four traces, ARTEMIS includes the bitrate of 2 Mbps in the OTL most of the time but not always. Finally, ARTEMIS never puts into the OTL the largest mega-manifest bitrate of 7 Mbps on all four traces and occasionally employs the second largest bitrate of 4.5 Mbps in the OTL. Overall, Figure 9 corroborates that ARTEMIS effectively adjusts its ladders for the current network conditions.

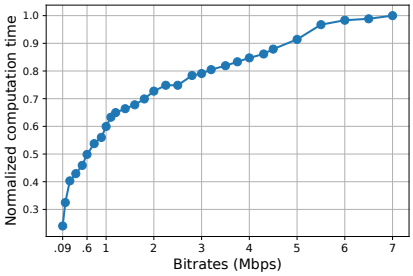

Figure 11: Normalized average computation time for encoding each bitrate in ARTEMIS.

Figure 10 evaluates ARTEMIS vs. the baselines in regard to computation cost, traffic volume, and ladder efficiency. While Figures 10a and 10b focus on the computation cost, we also investigate the relationship between the computation cost and processing time by measuring the time to encode each bitrate to the ultrafast preset on an EC2 machine and by calculating the average processing time to encode each segment according to the selected OTL. Figure 11 shows the normalized average computation time for encoding each bitrate. The findings corroborate earlier research conclusions that the computation cost is proportional to the processing time [29]. Figures 10a and 10b demonstrate that, by accounting for the network conditions, ARTEMIS' dynamic ladders provide lower computation cost than the baseline ladders, except for the Theo ladder that comprises

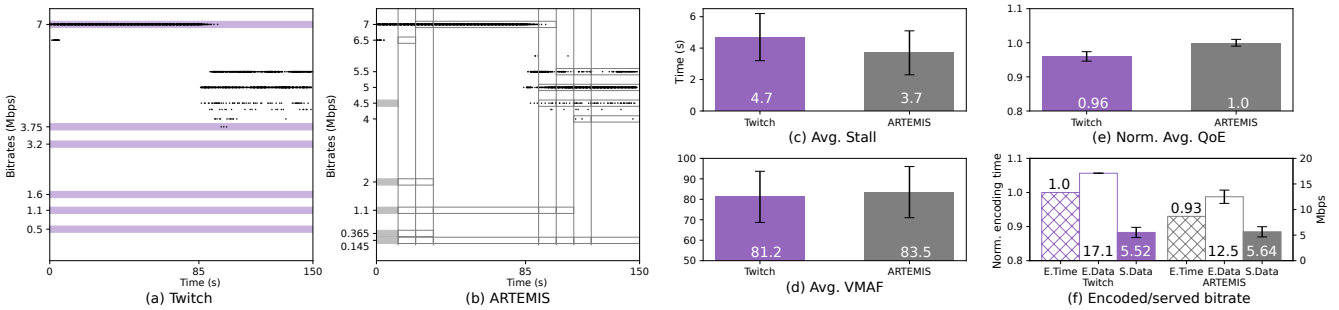

Figure 12: Requested and served bitrates, stall duration, VMAF, QoE, and encoding time with ARTEMIS' dynamic ladder vs. the static Twitch ladder when both approaches advertise the mega-manifest bitrates to the players.

only four representations. Despite limiting the maximum ladder length to five representations, ARTEMIS requires less computation than the Bitmovin ladder due to selecting a lower bitrate to cope with fluctuations in the available bandwidth. Additionally, ARTEMIS selects a smaller ladder when the bitrate preferences of the clients indicate a need in less than the maximum ℓ representations.

Figure 10c demonstrates that ARTEMIS' dynamic ladders increase the normalized average QoE compared to almost all baselines across the four network traces. This finding highlights the effectiveness of ARTEMIS even under challenging network conditions. While the Pensieve ladder on the LTE network trace delivers nearly the same QoE as ARTEMIS' dynamic ladders, the computation cost incurred by the Pensieve ladder is 44% higher than with ARTEMIS. Figure 10d shows that ARTEMIS substantially outperforms the static baselines with respect to ladder efficiency across all four network traces within a time slot of at most θ.

The network traces in our study exhibit high bandwidth fluctuations resulting in frequent stalls for the players. Whereas Figure 9 unveils that ARTEMIS reacts to the highly variable bandwidth by limiting the highest bitrate in the OTL to 4.5 Mbps, we conduct an additional experiment on a network trace with minimal bandwidth fluctuations. Specifically, the available bandwidth for each player is 10 Mbps initially, decreases to 6.5 Mbps after 85 s, and remains at this level for the subsequent 65 s. The experiment evaluates ARTEMIS' dynamic ladder against the static Twitch ladder, which is the only baseline that includes the largest considered bitrate of 7 Mbps. Additionally, we expose the mega-manifest bitrate preferences of the players not only to ARTEMIS but also to the Twitch alternative. In this setup, we anticipate ARTEMIS to serve the players with the highest available bitrate until 85 s into the experiment.

For this additional experiment, Figures 12a and 12b depict the bitrates requested by the players as black dots and the bitrates available for serving the requests as horizontal purple lines. The Twitch alternative supports the bitrate of 7 Mbps and affirmatively responds to all requests for this highest bitrate in the mega-manifest until 85 s into the experiment. After 85 s however, the players request bitrates that are below 7 Mbps and above 3.75 Mbps, which is the second highest bitrate in the Twitch ladder. Hence, the Twitch alternative serves the requests with the bitrate of 3.75 Mbps, which degrades the video quality by underutilizing the available bandwidth. On the other hand, Figure 12b shows that ARTEMIS efficiently selects the OTL to serve the requests with higher bitrates. The gray vertical lines indicate the times when ARTEMIS changes its OTL. During the first 10 s that comprise the first slot of the time-slotted operation, ARTEMIS uses the OTL predefined by the streaming service provider and depicted as gray rectangles in Figure 12b. Upon receiving requests for the bitrates of 6.5 and 7 Mbps during the first time slot, ARTEMIS updates the OTL and, due to low stalling and frequent requests for the bitrate of 7 Mbps, converges by the third time slot to serving the players with the bitrate of 7 Mbps until time 85 s. During the five consequent time slots, ARTEMIS instructs the live encoder to use the OTL with only three bitrates of 0.145, 1.1, and 7 Mbps, which efficiently reduces the computation and bandwidth costs. However, when the clients request lower bitrates at time 85 s, ARTEMIS reacts quickly, and Figure 12b shows that ARTEMIS determines the four bitrates that closely match all bitrates requested by the players.

Figures 12c-12f compare ARTEMIS with the Twitch alternative in regard to the average stall duration, VMAF, and normalized QoE as well as encoding time (labeled as E. Time), encoded bitrate (E. Data) and served bitrate (S. Data). The results show that ARTEMIS' dynamic ladders perform better than the static Twitch ladder in all metrics while reducing the traffic volume by 0.3%.

We further extend our exploration of Scenario I by evaluating QoE in Figure 16 of Appendix E, LoL$^+$ as an alternative low-latency ABR algorithm in Figure 17 of Appendix F, and influence of the content type in Figures 18 and 19 of Appendix G. We also study how the ARTEMIS performance depends on weight α in Figure 20 of Appendix H, maximum OTL length ℓ in Figures 21 and 22 of Appendix I, and time-slot duration θ in Figures 23 and 24 of Appendix J.

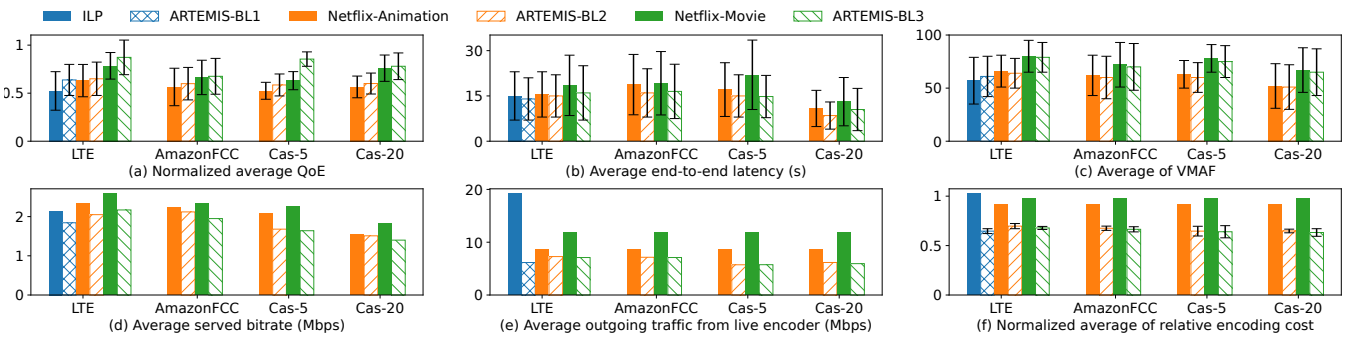

Figure 13: QoE, end-to-end latency, VMAF, served bitrates, traffic volume, and encoding cost for ARTEMIS' customized BL1, BL2, and BL3 ladders vs. the ILP, Netflix-Animation, and Netflix-Movie ladders.

Scenario II. We also assess ARTEMIS' dynamic ladders in comparison to the three fixed-length ladders described in Table 3 of Appendix D. These baselines aim at particular content types and/or available network bandwidth. Because the average bandwidth in the LTE network trace is nearly 3 Mbps, *i.e.*, as targeted by the ILP ladder, we evaluate ARTEMIS against the ILP ladder specifically on the LTE trace. However, we consider all available network traces for the two Netflix ladders. To cover the representations in the considered baselines, we customize ARTEMIS' mega-manifest by introducing additional representations into it. For example, the comparison with the ILP ladder customizes ARTEMIS' mega-manifest by including all representations of the ILP ladder and the additional seven representations described in Table 3.

Figure 13, along with Table 5 in Appendix K, shows that ARTEMIS consistently outperforms the baselines across all network traces in terms of QoE and end-to-end latency by selecting the maximum five representations. ARTEMIS also exhibits better performance in terms of stall duration, bitrate instability, and VMAF instability. While Figure 13c reveals a negligible decrease in VMAF, Figure 13d demonstrates that ARTEMIS effectively serves the players with lower bitrates, thereby reducing the traffic from the CDN to the players. Figure 13e illustrates that ARTEMIS decreases the volume of encoded data, reducing the consumption of computational resources in the live encoder and also lowering the traffic from the encoder to the CDN.

6 Conclusion

This paper presents ARTEMIS, a practical scalable system for efficient dynamic construction of effective bitrate ladders during a live session of video streaming. ARTEMIS seamlessly enhances the end-to-end HAS pipeline and innovatively leverages its standard features, such as the advertisement of representations via the mega-manifest to collect fine-grained client-side information transparently to heterogeneous ABR algorithms. The context-aware system is also efficient in its capture of the content complexity via PSNR. We provide a cloud-based implementation of ARTEMIS and extensively evaluate it in real-world and trace-driven experiments.

Whereas ARTEMIS pursues multifaceted goals that include low end-to-end latency, high QoE, and low encoding computation, the evaluation against prominent static bitrate ladders shows that ARTEMIS successfully delivers multi-objective improvements to reduce end-to-end latency by 18%, increase QoE by 11%, and decrease encoding computation by 25%. The deeper analysis of ARTEMIS' success reveals that its accounting for dynamic network conditions is crucial. When the available network bandwidth fluctuates significantly to increase the likelihood of stall events, ARTEMIS prioritizes lower bitrates in the ladder construction while maintaining QoE close to the maximum achievable under the poor network conditions. By scaling down the bitrate ladder, ARTEMIS also reduces encoding computation, decreases storage and bandwidth costs (especially at the network edge), and thereby enables the streaming service to support more clients. Under stable network conditions, ARTEMIS also lowers the resource consumption by composing a ladder with a smaller number of representations and improves QoE through a better alignment of the representations with the bitrate preferences of the clients.

Acknowledgments

We thank the anonymous NSDI 2024 reviewers and our shepherd Dave Oran for their invaluable feedback and guidance. The research is supported in part by the Austrian Federal Ministry for Digital and Economic Affairs, National Foundation for Research, Technology and Development, and Christian Doppler Research Association with project Christian Doppler Laboratory ATHENA (https://athena.itec.aau.at/) and the Spanish Ministry of Science and Innovation with grants TED2021-131264B-I00 (SocialProbing) and PID2021-128223OA-I00 (GreenEdge).

References

[1] Akamai, Content Delivery Network (CDN). [Online] Available: https://www.akamai.com/solutions/content-delivery-network.

[2] Amazon Elastic Compute Cloud (Amazon EC2). [Online] Available: https://aws.amazon.com/ec2/.

[3] Bitmovin and Akamai to Debut Joint CMCD Solution at NAB Show 2023. [Online] Available: https://bitmovin.com/press-room/akamai-bitmovin-create-joint-cmcd-solution/.

[4] Bitmovin Dashboard, Live Encoder. [Online] Available: https://bitmovin.com/dashboard/live.

[5] Bitmovin Live Encoder. [Online] Available: https://bitmovin.com/live-encoding-live-streaming/.

[6] Conviva. [Online] Available: https://www.conviva.com/technology/.

[7] CTA-5004: Web Application Video Ecosystem – Common Media Client Data. [Online] Available: https://cdn.cta.tech/cta/media/media/resources/standards/pdfs/cta-5004-final.pdf.

[8] Datazoom, Streaming Video Monitoring. [Online] Available: https://www.datazoom.io/streaming-video-monitoring/.

[9] Docker Hub. [Online] Available: https://hub.docker.com/.

[10] FFmpeg. [Online] Available: https://https://ffmpeg.org/.

[11] Mux Video. [Online] Available: https://docs.mux.com/guides/video/configure-broadcast-software#good---720p-30fps.

[12] Netflix Animation Bitrate Ladder. [Online] Available: https://ottverse.com/creating-the-perfect-encoding-ladder/,.

[13] Netflix Movie Bitrate Ladder. [Online] Available: https://ottverse.com/creating-the-perfect-encoding-ladder/,.

[14] Optimization with PuLP. [Online] Available: https://coin-or.github.io/pulp/.

[15] Raw Data — Measuring Broadband America. [Online] Available: https://www.fcc.gov/reportsresearch/reports.

[16] Theo Player. [Online] Available: https://www.theoplayer.com/blog/encoding-cost-efficient-streaming.

[17] Twitch. [Online] Available: https://stream.twitch.tv/encoding/.

[18] Hadi Amirpour, Raimund Schatz, and Christian Timmerer. Between Two and Six? Towards Correct Estimation of JND Step Sizes for VMAF-Based Bitrate Laddering. In *QoMEX*, pages 1–4, September 2022.

[19] Hadi Amirpour, Christian Timmerer, and Mohammad Ghanbari. PSTR: Per-Title Encoding Using Spatio-Temporal Resolutions. In *IEEE ICME*, pages 1–6, July 2021.

[20] Apple. Enabling Low-Latency HTTP Live Streaming (HLS). [Online] Available: https://developer.apple.com/documentation/http_live_streaming/enabling_low-latency_http_live_streaming_hls, 2021. Accessed on Jan. 26, 2023.

[21] Apple Inc. HTTP Live Streaming (HLS) Authoring Specification for Apple Devices. [Online] Available: http://bit.ly/hls_spec_2017, 2015. Online; accessed on Jan. 05, 2022.

[22] Abdelhak Bentaleb, Mehmet N. Akcay, May Lim, Ali C. Begen, and Roger Zimmermann. Catching the Moment with LoL+ in Twitch-like Low-Latency Live Streaming Platforms. *IEEE Transactions on Multimedia*, 24:2300–2314, 2021.

[23] Abdelhak Bentaleb, May Lim, Mehmet N. Akcay, Ali C. Begen, and Roger Zimmermann. Common Media Client Data (CMCD): Initial Findings. In *NOSSDAV*, pages 25–33, July 2021.

[24] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP. *IEEE Communications Surveys & Tutorials*, 21(1):562–585, 2019.

[25] Madhukar Bhat, Jean-Marc Thiesse, and Patrick Le Callet. Combining Video Quality Metrics To Select Perceptually Accurate Resolution In A Wide Quality Range: A Case Study. In *IEEE ICIP*, pages 2164–2168, September 2021.

[26] DASH-IF. Low-Latency Modes for DASH. [Online] Available: https://dashif.org/docs/CR-Low-Latency-Live-r8.pdf, 2020. Accessed on Jan. 26, 2023.

[27] DASH-IF. DASH Reference Player (dash.js). [Online] Available: https://reference.dashif.org/dash.js/, 2021. Online; accessed on Jan. 22, 2022.

[28] Jan De Cock, Zhi Li, Megha Manohara, and Anne Aaron. Complexity-Based Consistent-Quality Encoding in the Cloud. In *IEEE ICIP*, pages 1484–1488, September 2016.

[29] Alireza Erfanian, Hadi Amirpour, Farzad Tashtarian, Christian Timmerer, and Hermann Hellwagner. LwTE: Light-Weight Transcoding at the Edge. *IEEE Access*, 9:112276–112289, 2021.

[30] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. 1990.

[31] Syed Hasan, Sergey Gorinsky, Constantine Dovrolis, and Ramesh K. Sitaraman. Trade-Offs in Optimizing the Cache Deployments of CDNs. In *IEEE INFOCOM*, pages 460–468. IEEE, 2014.

[32] Tianchi Huang, Rui-Xiao Zhang, and Lifeng Sun. Deep Reinforced Bitrate Ladders for Adaptive Video Streaming. In *NOSSDAV*, pages 66–73, 2021.

[33] Tianchi Huang, Chao Zhou, Rui-Xiao Zhang, Chenglei Wu, Xin Yao, and Lifeng Sun. Comyco: Quality-Aware Adaptive Video Streaming via Imitation Learning. In *ACM MM*, pages 429–437, 2019.

[34] Quan Huynh-Thu and Mohammad Ghanbari. Scope of Validity of PSNR in Image/Video Quality Assessment. *Electronics Letters*, 44:800–801(1), June 2008.

[35] ISO/IEC. ISO/IEC 23000-19:2020 Information Technology – Multimedia Application Format (MPEG-A) – Part 19: Common Media Application Format (CMAF) for Segmented Media. [Online] Available: https://www.iso.org/standard/79106.html, 2020. Accessed on Jan. 26, 2023.

[36] ISO/IEC. 2019. Information Technology — Dynamic Adaptive Streaming over HTTP (DASH) — Part 1: Media Presentation Description and Segment Formats. International standard 23009-1:2019, International Organization for Standardization, December 2019.

[37] Jan Ozer. Saving on H.264 Encoding and Streaming: Deploy Capped CRF. [Online] Available: https://tinyurl.com/CappedCRF, 2018. Online; accessed on Feb. 12, 2022.

[38] Alan B. Johnston and Daniel C. Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, 2012.

[39] Theo Karagkioules, Rufael Mekuria, Dirk Griffioen, and Arjen Wagenaar. Online Learning for Low-Latency Adaptive Streaming. In *ACM MMSys*, pages 315–320, 2020.

[40] Angeliki V. Katsenou, Fan Zhang, Kyle Swanson, Mariana Afonso, Joel Sole, and David R. Bull. VMAF-Based Bitrate Ladder Estimation for Adaptive Streaming. In *PCS*, pages 1–5, June 2021.

[41] Dilip Kumar Krishnappa, Michael Zink, and Ramesh K. Sitaraman. Optimizing the Video Transcoding Workflow in Content Delivery Networks. In *ACM MMSys*, pages 37–48, 2015.

[42] Pierre Lebreton and Kazuhisa Yamagishi. Predicting User Quitting Ratio in Adaptive Bitrate Video Streaming. *IEEE Transactions on Multimedia*, 23:4526–4540, 2021.

[43] Pierre Lebreton and Kazuhisa Yamagishi. Quitting Ratio-Based Bitrate Ladder Selection Mechanism for Adaptive Bitrate Video Streaming. *IEEE Transactions on Multimedia*, pages 1–14, 2023.

[44] Dayoung Lee, Jungwoo Lee, and Minseok Song. Video Quality Adaptation for Limiting Transcoding Energy Consumption in Video Servers. *IEEE Access*, 7:126253–126264, 2019.

[45] Zhi Li, Christos Bampis, Julie Novak, Anne Aaron, Kyle Swanson, Anush Moorthy, and Jan De Cock. VMAF: The Journey Continues. *Netflix Technology Blog*, 25(1), 2018.

[46] Melissa Licciardello, Lukas Humbel, Fabian Rohr, Maximilian Grüner, and Ankit Singla. [Solution] Prepare Your Video for Streaming with Segue. *Journal of Systems Research*, 2(1), 2022.

[47] Megha Manohara, Anush Moorthy, Jan De Cock, Ioannis Katsavounidis, and Anne Aaron. Optimized Shot-Based Encodes: Now Streaming. *The Netflix Tech Blog*, 2018.

[48] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*, pages 197–210, 2017.

[49] Vignesh V. Menon, Hadi Amirpour, Mohammad Ghanbari, and Christian Timmerer. ETPS: Efficient Two-Pass Encoding Scheme for Adaptive Live Streaming. In *IEEE ICIP*, pages 1516–1520, October 2022.

[50] Vignesh V. Menon, Hadi Amirpour, Mohammad Ghanbari, and Christian Timmerer. OPTE: Online Per-Title Encoding for Live Video Streaming. In *IEEE ICASSP*, pages 1865–1869, May 2022.

[51] Jan Ozer. Encoding Ladders: What You Need to Know. https://www.wowza.com/blog/encoding-ladders-what-you-need-to-know, 2022. [Online; accessed 27-April-2023].

[52] Roger Pantos and William May. HTTP Live Streaming. RFC 8216, August 2017.

[53] Yuriy A. Reznik, Xiangbo Li, Karl O. Lillevold, Abhijith Jagannath, and Justin Greer. Optimal Multi-Codec Adaptive Bitrate Streaming. In *IEEE ICMEW*, pages 348–353, 2019.

[54] Yuriy A. Reznik, Karl O. Lillevold, Abhijith Jagannath, Justin Greer, and Jon Corley. Optimal Design of Encoding Profiles for ABR Streaming. In *PV Workshop*, pages 43–47, June 2018.

[55] Sandvine Inc. The Global Internet Phenomena Report. [Online] Available: https://www.sandvine.com/phenomena, 2022. Online; accessed on March. 15, 2022.

[56] Babak Taraghi, Minh Nguyen, Hadi Amirpour, and Christian Timmerer. Intense: In-Depth Studies on Stall Events and Quality Switches and Their Impact on the Quality of Experience in HTTP Adaptive Streaming. *IEEE Access*, 9:118087–118098, 2021.

[57] Babak Taraghi, Anatoliy Zabrovskiy, Christian Timmerer, and Hermann Hellwagner. CAdViSE: Cloud-Based Adaptive Video Streaming evaluation Framework for the Automated Testing of Media Players. In *ACM MMSys*, pages 349–352, 2020.

[58] Farzad Tashtarian, Abdelhak Bentaleb, Hadi Amirpour, Babak Taraghi, Christian Timmerer, Hermann Hellwagner, and Roger Zimmermann. LALISA: Adaptive Bitrate Ladder Optimization in HTTP-Based Adaptive Live Streaming. In *IEEE/IFIP NOMS*, 2023.

[59] Farzad Tashtarian, Abdelhak Bentaleb, Alireza Erfanian, Hermann Hellwagner, Christian Timmerer, and Roger Zimmermann. HxL3: Optimized Delivery Architecture for HTTP Low-Latency Live Streaming. *IEEE Transactions on Multimedia*, 25:2585–2600, 2022.

[60] Laura Toni, Ramon Aparicio-Pardo, Karine Pires, Gwendal Simon, Alberto Blanc, and Pascal Frossard. Optimal Selection of Adaptive Streaming Representations. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 11(2s):1–26, February 2015.

[61] Jeroen Van der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters*, 20(11):2177–2180, 2016.

[62] Abhinau K. Venkataramanan, Cosmin Stejerean, and Alan C. Bovik. FUNQUE: Fusion of Unified Quality Evaluators. In *IEEE ICIP*, pages 2147–2151, October 2022.

[63] Wowza Media Systems. Video Streaming Latency Report. September 2019, Report, https://www.wowza.com/wp-content/uploads/Streaming-Video-Latency-Report-Interactive-2019.pdf.

[64] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity. In *NSDI*, pages 303–320, 2021.

[65] Hongcheng Zhong, Jun Xu, Chen Zhu, Donghui Feng, and Li Song. Complexity-Oriented Per-Shot Video Coding Optimization. In *IEEE ICME*, pages 1–6, 2022.

A QoE Model

Our work and, specifically, the evaluation metrics in Section 5.2 adopt the following QoE model to express QoE as a weighted sum of four terms [33]:

$$QoE = \omega_1 \sum_{n=1}^{N} q(R_n) + \omega_2 \sum_{n=1}^{N} T_n + \omega_3 \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_+ + \omega_4 \sum_{n=1}^{N-1} [q(R_{n+1}) - q(R_n)]_- \quad (11)$$

where bitrate R_n and stall time T_n characterize each segment n, function $q(R_n)$ computes VMAF of the segment, and the last two terms express VMAF instability over N consecutive segments, with different weights for VMAF increases and decreases. The QoE model uses weights $\omega_1 = 0.8469$, $\omega_2 = -28.7959$, $\omega_3 = 0.2979$, and $\omega_4 = -1.0610$.

B Correlation between PSNR and VMAF

To measure video quality quickly enough for live streaming, the ARTEMIS design in Section 4.2 relies on PSNR rather than VMAF. For different segment durations and content types, Figure 14 demonstrates that PSNR strongly correlates with VMAF.

C Notation in the OTL Selection

Table 2 sums up the notation in Section 4.3 that formulates and solves the MILP to determine a new bitrate ladder.

D Experimental Settings

To elaborate on the setup presented in Section 5.2, Table 3 describes the three static content-aware ILP, Netflix-Animation, and Netflix-Movie bitrate ladders in contrast with three customized ARTEMIS ladders. Figure 15 illustrates bandwidth fluctuations in the LTE, AmazonFCC, Cascade-5, and Cascade-20 network traces used in our experiments.

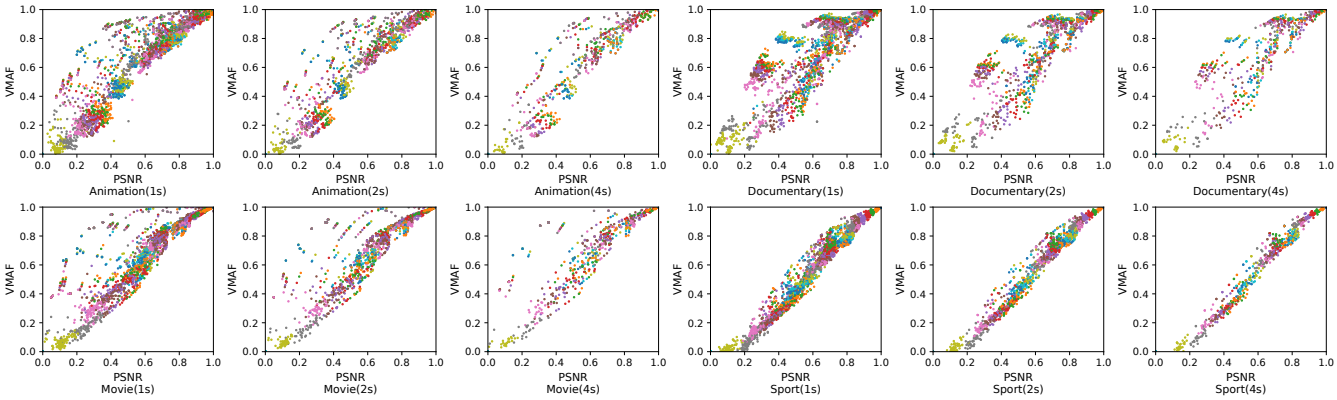

Figure 14: Correlation between PSNR and VMAF.

Table 2: Notation summary.

Symbol	*Description*
	Input parameters
B	Set with indexes of the m representations in the mega-manifest
b_i	Bitrate of representation i
r_i	Number of requests for bitrate b_i
$F(b_i)$	Quality indicator function that predicts the PSNR value of the segment with bitrate b_i
ℓ	Maximum number of representations in the OTL
β	Maximum changes between two successive OTLs
α	Weight in the objective function
$\bar{x}_i$	Indicator whether bitrate b_i appears in the previous OTL
	Variables
x_i	Indicator whether bitrate b_i appears in the the new OTL
$y_{j,i}$	Indicator whether the OTL uses bitrate b_j to accommodate the requests for higher bitrate b_i
q	Quality improvement that the requested bitrates would provide compared to the OTL bitrates
s	Traffic reduction due to using the OTL bitrates instead of the requested bitrates

E Additional Results for Scenario I

To complement the experimental results presented for Scenario I in Figures 8a-8e of Section 5.3, Table 4 depicts the mean and standard deviation for the stall duration, bitrate, and VMAF. The results on the stall duration show that ARTEMIS outperforms the baselines by efficiently updating the OTL. Due to leveraging the client-side information, ARTEMIS succeeds in detecting the stalls and adjusts the OTL by selecting lower bitrates to mitigate the stalls while maintaining an acceptable average VMAF value. Table 4 reveals that ARTEMIS serves the players with lower average bitrates for almost all network traces and numbers of players. However, because ARTEMIS considers the estimated PSNR values of the subsequent segments, the lower served bitrates decrease VMAF insignificantly. The comparison with the Theo ladder on the LTE network trace shows that ARTEMIS even improves the average VMAF in some situations despite decreasing the average served bitrates. Figure 16 evaluates the normalized average QoE with ARTEMIS' dynamic ladders vs. the five static baselines.

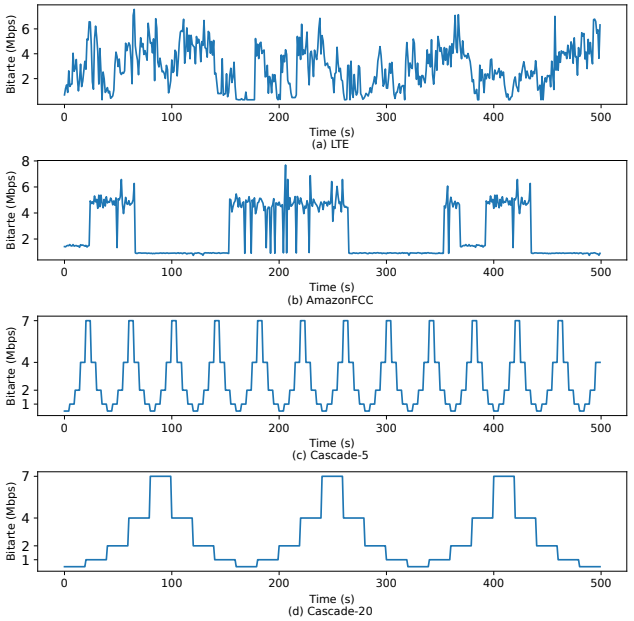

Figure 15: Bandwidth fluctuations in the network traces.

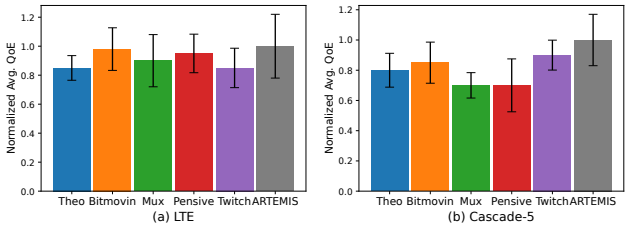

Figure 16: Normalized average QoE with ARTEMIS' dynamic ladders vs. the five static baselines.

Table 3: Representations in the three static content-aware ILP, Netflix-Animation, and Netflix-Movie ladders vs. three customized ARTEMIS ladders.

BL \ Res.@Mbps	224p@0.052	360p@0.082	360p@0.283	720p@0.451	720p@1.0	720p@1.4	720p@1.625	720p@1.8	720p@2.25	720p@3.002	720p@3.4	720p@3.75	720p@4.3	72pp@5.32	1080p@8.34
ILP [60]	✓	✓	✓	✓			✓			✓				✓	✓
ARTEMIS-BL1 [60]	✓	✓	✓	✓	+	+	✓	+	+	✓	+	+	+	✓	✓

	360p@0.2	440p@0.333	640p@0.533	630p@0.725	630p@1.0	810p@1.275	810p@1.4	810p@1.6	1080p@1.8	1080p@2.1	1080p@2.25	1080p@2.8	1080p@3.2	1080p@3.5	
Netflix-Animation [12]	✓	✓	✓	✓		✓				✓				✓	
ARTEMIS-BL2	✓	✓	✓	✓	+	✓	+	+	+	✓	+	+	+	✓	

	315p@0.2	503p@0.4	743p@0.638	743p@0.75	743p@1.088	900p@1.775	900p@2.0	900p@2.25	900p@2.5	1013p@2.95	1013p@3.4	1013p@3.75	1013p@4.3	1080p@4.875	
Netflix-Movie [13]	✓	✓	✓		✓	✓				✓				✓	
ARTEMIS-BL3	✓	✓	✓	+	✓	✓	+	+	+	✓	+	+	+	✓	

Table 4: Impact by the Theo (T), ARTEMIS (A), Bitmovin (B), Mux (M), Pensieve (P), and Twitch (W) bitrate ladders (BLs) on the stall duration, bitrate, and VMAF of served segments on four network traces (NTs).

NT	BL	Stall duration (s)				Bitrate (Mbps)				VMAF			
		#10	#20	#50	#100	#10	#20	#50	#100	#10	#20	#50	#100
LTE	T	37.1±13.9	41.3±17.4	37.5±14.8	36.7±13.1	2.2±0.5	2.3±0.5	2.1±0.5	2.2±0.6	57.0±21.0	58.5±20.5	57.5±20.9	57.9±20.8
	A	36.4±11.6	29.9±10.6	30.6±9.9	33.5±10.4	2.0±0.4	1.7±0.3	1.7±0.2	2.1±0.4	60.0±18.5	60.4±18.5	60.6±18.4	62.5±18.9
	B	36.1±15.9	38.6±14.1	37.1±11.3	34.3±13.4	2.2±0.6	2.1±0.5	2.1±0.5	2.1±0.5	62.6±19.4	61.7±18.7	62.2±18.9	62.3±19.2
	M	44.0±15.4	42.2±14.1	47.3±11.3	43.7±13.4	2.4±0.6	2.3±0.5	2.5±0.6	2.5±0.6	62.5±19.1	62.2±18.3	64.2±17.9	63.4±18.4
	P	45.0±16.6	42.4±14.8	40.5±14.2	39.4±13.0	2.6±0.7	2.5±0.6	2.5±0.6	2.5±0.6	64.8±17.6	64.0±17.7	65.0±17.8	64.8±17.4
	W	41.9±14.6	43.8±17.7	42.1±12.2	43.7±15.7	2.3±0.6	2.4±0.8	2.4±0.8	2.5±0.8	62.1±18.4	63.1±17.5	62.5±17.6	63.2±17.8
AmazonFCC	T	40.7±14.8	44.0±16.1	40.2±15.2	40.8±17.5	2.0±0.7	2.1±0.7	2.0±0.7	2.0±0.8	53.9±22.6	55.1±22.2	55.8±23.2	55.4±22.3
	A	39.0±16.4	35.9±14.4	35.1±15.2	35.0±14.3	1.5±0.6	1.5±0.5	1.7±0.5	1.7±0.6	51.3±22.1	50.9±22.8	53.2±22.0	53.5±22.2
	B	44.1±20.0	39.6±21.8	35.7±19.1	36.1±20.2	1.9±0.8	1.8±0.7	1.8±0.7	1.8±0.7	56.2±22.1	55.1±23.1	54.0±23.5	55.2±22.7
	M	62.0±13.9	63.1±17.9	53.3±18.5	55.2±19.7	2.5±0.7	2.5±0.6	2.3±0.6	2.3±0.7	61.4±20.8	62.4±19.3	54.3±21.4	55.4±20.9
	P	51.3±19.0	46.7±18.2	53.5±17.2	42.2±18.6	2.2±0.7	2.2±0.8	2.2±0.8	2.2±0.7	58.3±21.9	57.7±21.2	57.2±22.1	58.0±21.7
	W	52.1±20.9	50.2±21.4	48.5±19.2	48.9±20.7	2.4±1.2	2.5±1.2	2.3±1.2	2.3±1.1	58.5±20.9	59.1±20.8	57.3±21.0	58.1±21.3
Cascade-5	T	37.1±10.9	36.6±10.8	32.6±8.8	32.0±11	1.8±0.3	1.7±0.4	1.8±0.3	1.7±0.3	52.7±22.5	51.0±22.5	52.1±23.4	51.8±22.4
	A	29.6±8.9	30.7±10.3	32.6±11.2	27.7±10.3	1.5±0.2	1.6±0.2	1.6±0.3	1.5±0.3	56.5±20.2	55.3±20.6	56.7±20.8	54.5±21.7
	B	36.0±10.1	36.3±9.5	35.7±10.5	33.6±12.5	1.8±0.3	1.7±0.3	1.8±0.4	1.7±0.3	57.4±21.9	56.0±22.3	56.4±22.0	56.3±21.6
	M	45.1±10.6	48.1±10.3	46.8±10.6	44.7±10.3	2.1±0.3	2.0±0.3	2.0±0.3	2.0±0.3	48.4±21.3	57.7±21.2	57.4±20.9	57.2±21.1
	P	43.5±11.2	46.0±12.0	44.2±9.6	42.5±12.6	2.1±0.4	2.1±0.4	2.1±0.3	2.0±0.4	58.6±21.2	58.8±19.9	59.1±20.4	57.9±20.6
	W	37.3±10.7	43.3±12.0	38.5±11.3	38.4±12.9	1.9±0.3	2.0±0.4	1.8±0.3	1.9±0.4	57.2±20.5	57.4±20.5	56.4±21.0	57.1±20.4
Cascade-20	T	26.4±15.7	21.8±12.4	19.3±13.7	20.5±14.5	1.8±0.2	1.6±0.3	1.6±0.2	1.6±0.3	54.1±22.4	51.3±23.6	51.2±23.8	49.0±23.0
	A	11.4±8.2	13.3±11.9	16.7±10.5	15.1±11.2	1.7±0.4	1.5±0.2	1.4±0.3	1.4±0.3	55.2±22.7	52.5±25.3	51.1±23.3	50.3±21.8
	B	15.4±10.9	18.8±13.3	17.3±14.2	17.4±13.9	1.6±0.1	1.6±0.3	1.6±0.3	1.6±0.3	53.1±23.9	52.5±25.2	52.2±24.9	51.9±25.0
	M	30.0±15.1	34.5±13.7	32.3±14.9	32.0±15.0	2.0±0.2	2.0±0.2	2.0±0.3	2.0±0.3	55.2±21.9	55.3±22.3	56.9±23.3	54.0±23.0
	P	20.4±13.4	22.5±14.6	20.8±14.5	21.0±14.1	1.8±0.2	1.8±0.3	1.7±0.2	1.7±0.3	55.0±22.6	54.7±23.7	54.7±23.5	52.0±24.0
	W	23.9±15.0	26.5±14.5	24.8±15.2	25.2±15.5	1.8±0.2	1.8±0.3	1.8±0.3	1.8±0.2	54.7±22.8	54.2±23.4	54.6±23.8	51.9±23.0

F Sensitivity to the ABR Algorithm

While the default ABR algorithm in our experiments is L2A, we also examine how the choice of the ABR algorithm affects the ARTEMIS performance. Specifically, we consider LoL$^+$ as an alternative to L2A. Figure 17 evaluates ARTEMIS' dynamic ladders against the five static baselines on the four network traces. The results demonstrate that ARTEMIS consistently outperforms the baselines in terms of the stall duration, bitrate switching, VMAF switching, bitrate instability, and VMAF instability regardless of the chosen ABR algorithm. Although Figure 17b shows that ARTEMIS serves the players with lower bitrates, Figures 17g and 17h reveal that the decreases in VMAF and PSNR are negligible.

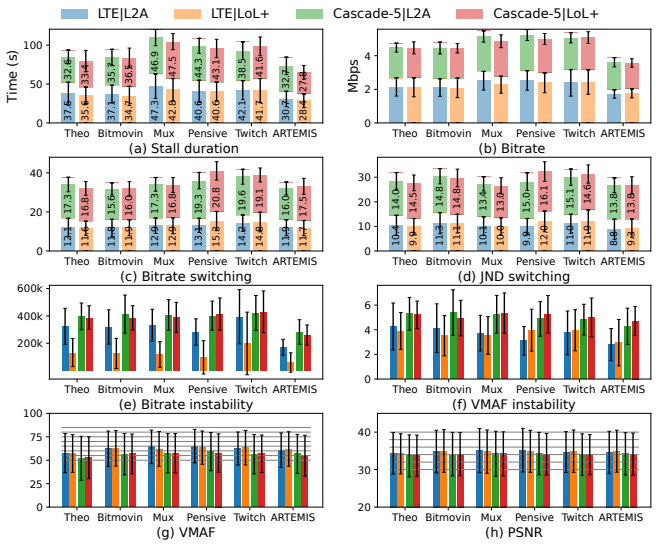

Figure 17: Impact of choosing L2A vs. LoL+ as the ABR algorithm on the performance with the five static ladders and ARTEMIS' dynamic ladders.

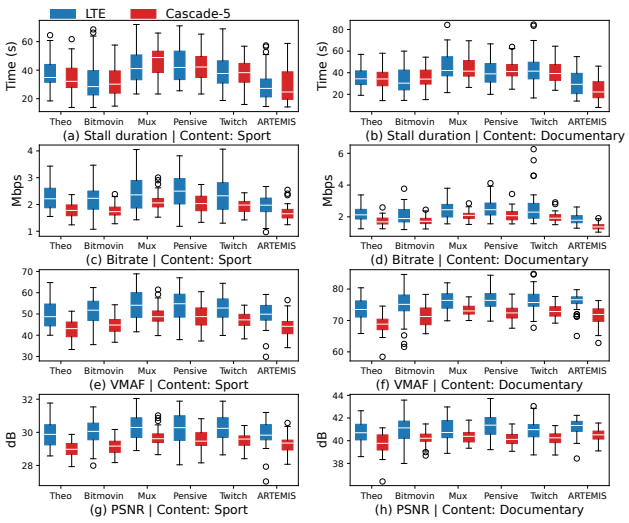

Figure 18: Stall duration, served bitrates, VMAF, and PSNR under the five static ladders and ARTEMIS' dynamic ladders for sport vs. documentary as the content type.

G Influence of the Content Type

We assess ARTEMIS' dynamic ladders against the five static baselines on the LTE and Cascade-5 network traces. Figure 18 reports on the stall duration, bitrates, VMAF, and PSNR. ARTEMIS reduces the stall reduction compared to the static fixed-length baselines for two different content types. We attribute the reduction to the substantial bandwidth fluctuations in both considered network traces, which leads to frequent stalls. Figures 18c and 18d illustrate that ARTEMIS mitigates the stalls by adding representations with lower bitrates to the OTL. Despite the reduction in the served bitrates, ARTEMIS maintains an acceptable VMAF value for the two content types. ARTEMIS surpasses the baseline performance because its ladder construction accounts for video quality measured via PSNR which, unlike VMAF, is quickly computable. The comparison of Figures 18c and 18d with Figures 18g and 18h suggests a strong correlation between PSNR and VMAF. Figure 14 in Appendix B corroborates this correlation.

Figure 19 extends the evaluation by reporting on the bitrate switching, bitrate instability, JND switching, and VMAF instability. We set JND to 6 [18]. While the bitrate switching is similar with ARTEMIS' dynamic ladders and baselines, ARTEMIS reduces the bitrate instability compared to the baseline ladders. We attribute this reduction to composing the mega-manifest from the large number of representations. Overall, the evaluation shows that ARTEMIS outperforms the baselines with different content types.

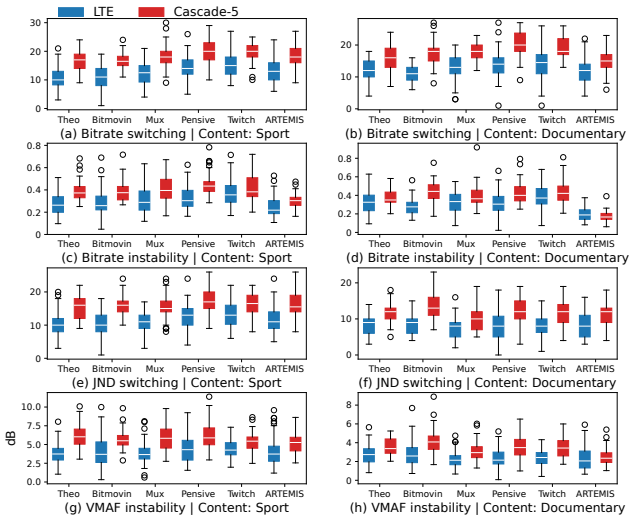

Figure 19: Bitrate switching, bitrate instability, JND switching, and VMAF instability under the five static ladders and ARTEMIS' dynamic ladders for sport vs. documentary as the content type.

H Impact of Weight α

We evaluate the impact of weight α on the LTE trace and a constant-bandwidth trace where the network bandwidth available for each player remains fixed at 7 Mbps throughout the streaming session. We assess how ARTEMIS performs when provided with the following three *StallAlpha* dictionaries:

- D-I = {1:[0,1], 0.9:[1,2], 0.8:[2,3], 0.7:[3,4], 0.6:[4,5], 0.5:[5,100]},
- D-II = {1:[0,2], 0.9:[2,4], 0.8:[4,6], 0.7:[6,8], 0.6:[8,10], 0.5:[10,100]}, and

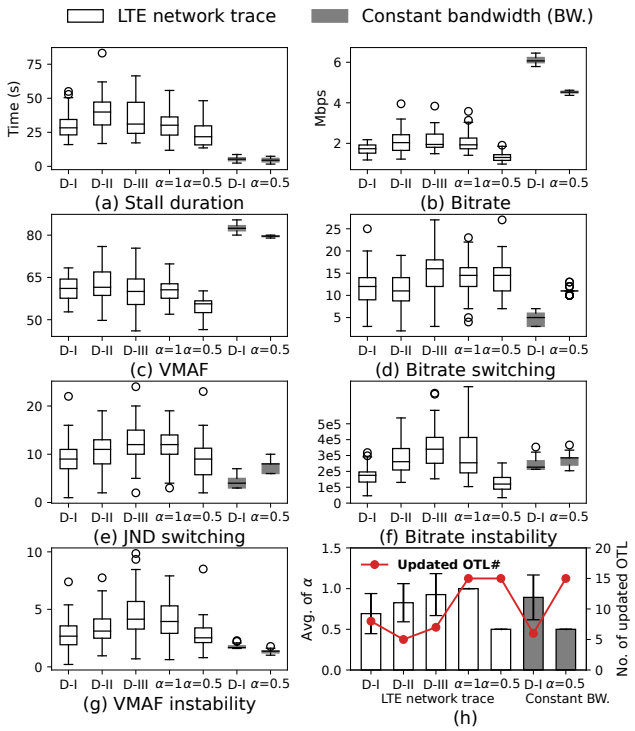

Figure 20: Impact of using the D-I, D-II, and D-III dictionaries to dynamically determine the α value vs. employing the static α values of 0.5 and 1.

- D-III = {1:[0,5], 0.9:[5,10], 0.8:[10,15], 0.7:[15,100]}.

The D-I, D-II, and D-III *StallAlpha* dictionaries exhibit different sensitivity levels to stall events. Dictionary D-I prioritizes stall duration over video quality and decreases α in response to short stalls. For example, ARTEMIS with dictionary D-I sets α to 1 when the average stall duration in the current time slot, which lasts 10 s, is less than 1 s. On the other hand, dictionary D-III sets α to 1 when the average stall duration is below 5 s. Figure 20 shows that reduction of this sensitivity causes more stalls and increases the bitrate instability, JND switching, bitrate instability, and VMAF instability.

We also consider an ARTEMIS variant that uses a static α value instead of relying on the *StallAnalysis*() function to determine α dynamically. Figure 20 indicates that the ARTEMIS instances with dictionary D-I and constant α of 0.5 behave similarly on the LTE trace. The similar behavior occurs because the bandwidth in the LTE trace fluctuates significantly and forces ARTEMIS with dictionary D-I to update the OTL eight times in order to handle stalls. Figure 20h demonstrates that the average of dynamic α values for ARTEMIS with dictionary D-I is 0.69, which is close to 0.5 in the static α setting. However, a static α value makes ARTEMIS perform suboptimally on the constant-bandwidth network trace by considerably reducing the video quality. Based on the experiments, we select D-I as the default

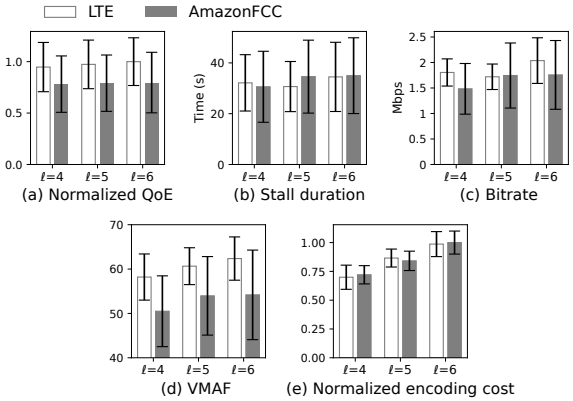

Figure 21: Impact of maximum OTL length ℓ on the ARTEMIS performance.

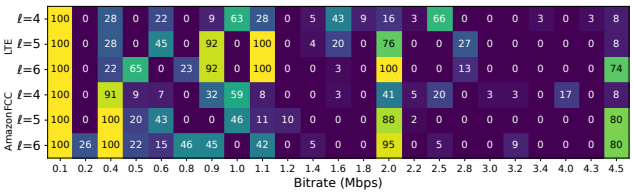

Figure 22: Bitrates selected by ARTEMIS with different values of maximum OTL length ℓ.

StallAlpha dictionary of ARTEMIS because dictionary D-I supports the best response to different network conditions while maintaining high video quality.

I Impact of Maximum OTL Length ℓ

We run ARTEMIS on the LTE and AmazonFCC network traces with different values of maximum OTL length ℓ. Figure 21 presents an exciting result that an increase in ℓ improves the served bitrates, VMAF, and encoding cost. However, these improvements do not guarantee higher QoE, *i.e.*, longer OTLs increase the resource cost without achieving a remarkable improvement in QoE. With relatively short OTLs, the difference between two adjacent bitrates in the OTL is typically large. Thus, the ABR algorithm keeps requesting the same bitrate when the available network bandwidth fluctuates mildly. On the other hand, when the bitrate switching does occur, the amplitude of the bitrate change is high. With a longer OTL, the bitrate switching becomes more frequent, and the bitrate-change amplitude diminishes. Figure 22 depicts the bitrates selected by ARTEMIS for its OTL with different values of the maximum OTL length.

J Impact of Time-Slot Duration θ

We investigate the impact of the time-slot duration on the ARTEMIS performance. It is essential for the time-slot duration to be neither too long nor too short. When the

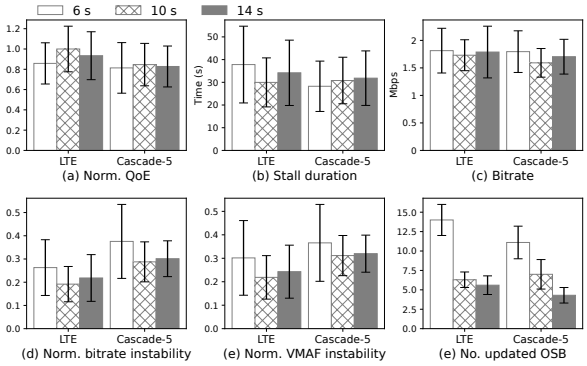

Figure 23: Impact of the time-slot duration on the ARTEMIS performance with the segment duration of 2 s.

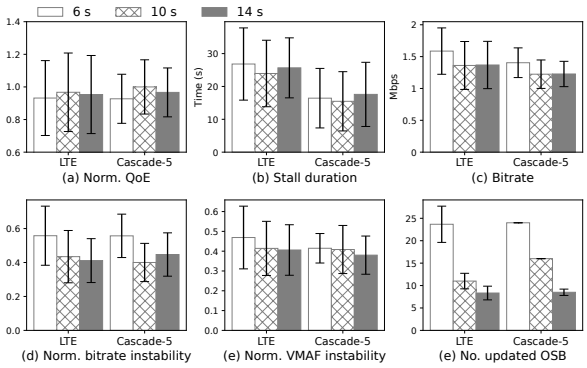

Figure 24: Impact of the time-slot duration on the ARTEMIS performance with the segment duration of 1 s.

time-slot duration is too long, ARTEMIS might be unable to promptly react to condition changes because ARTEMIS checks for a need to update the OTL only at the end of each time slot. The slow response degrades QoE. On the other hand, if the time-slot duration is too short, ARTEMIS incurs larger overhead, *e.g.*, due to solving the MILP more frequently. Other potential concerns include unnecessary updates of the OTL, higher bitrate instability for the clients, drifts in the live encoder, and emergence of video compression artifacts due to frequent changes in the OTL.

Figures 23 and 24 evaluate the impact of the time-slot duration on the ARTEMIS performance when the segment duration is set to 2 s or 1 s, respectively. The experiments rely on the LTE and Cascade-5 network traces and consider 6, 10, and 14 s as three values of the time-slot duration. For both values of the segment duration, our results show that the time-slot duration of 10 s provides the best performance. An algorithm for dynamic configuration of the time-slot duration is an exciting direction for future research on ARTEMIS.

Table 5: Stall duration, bitrate instability, and VMAF instability under ARTEMIS' dynamic ladders compared to the ILP and Netflix ladders in Scenario II.

NT	Bitrate ladder	Stall (s)	Bitrate instability (×1e6)	VMAF instability
LTE	ILP BL [60]	32.3±14.6	0.5±0.2	6.3±2.8
	ARTEMIS-BL1 [60]	31.1±12.7	0.2±0.08	3.2±1.4
	Netflix-Animation	35.5±11.8	0.27±0.1	3.9±1.8
	ARTEMIS-BL2	33.2±11.1	0.19±0.07	3.6±2.1
	Netflix-Movie	40.6±13.5	0.37±0.13	4.0±2.1
	ARTEMIS-BL3	36.1±16.6	0.22±0.077	3.0±1.6
AmazonFCC	Netflix-Animation	36.5±12.7	0.21±0.032	3.29±1.4
	ARTEMIS-BL2	34.5±14.4	0.18±0.069	3.22±1.6
	Netflix-Movie	40.8±13.8	0.35±0.12	4.8±2.2
	ARTEMIS-BL3	35.1±10.11	0.20±0.069	4.5±2.3
Cascade-5	Netflix-Animation	37.2±9.0	0.296±0.065	4.5±1.41
	ARTEMIS-BL2	29.5±9.1	0.185±0.041	3.4±1.07
	Netflix-Movie	45.5±0.2	0.42±0.087	4.5±1.5
	ARTEMIS-BL3	31.5±12.3	0.22±0.081	3.5±1.4
Cascade-20	Netflix-Animation	20.7±11.2	0.1±0.026	2.4±0.5
	ARTEMIS-BL2	17.1±13.2	0.1±0.038	2.1±0.7
	Netflix-Movie	24.5±11.6	0.16±0.035	3.2±1.09
	ARTEMIS-BL3	17.9±13.8	0.13±0.032	2.9±0.92

K Extra Results for Scenario II

Table 5 characterizes the performance of the ILP, Netflix-Animation, and Netflix-Movie bitrate ladders compared to ARTEMIS' three customized ladders in Scenario II.