This is the Pre-Published Version.

The following publication G. Zhang, J. Y. B. Lee, K. Liu, H. Hu and V. Aggarwal, "A Unified Framework for Flexible Playback Latency Control in Live Video Streaming," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 12, pp. 3024-3037, Dec. 2021 is available at https://doi.org/10.1109/TPDS.2021.3083202.

A Unified Framework for Flexible Playback Latency Control in Live Video Streaming

Guanghui Zhang, Jack Y. B. Lee, *Senior Member*, *IEEE*, Ke Liu,
Haibo Hu, *Senior Member*, *IEEE*, and Vaneet Aggarwal, *Senior Member*, *IEEE*

Abstract—Live video streaming is rapidly becoming a mainstream application in the mobile Internet. An important fact in live streaming is that the demand for low playback-latency inherently conflicts with the desire for high QoE. This requires different types of live services to seek different latency-QoE tradeoffs according to their service-requirements. However, our investigations revealed that it is fundamentally difficult for existing streaming algorithms to keep consistent latency across network conditions, let alone achieve the service-desired latency-QoE tradeoff. To tackle the challenge, this work develops a novel framework called Flexible Latency Aware Streaming (FLAS) that not only achieves consistent low latency, but also can control the latency-QoE tradeoff flexibly. Specifically, FLAS generates a set of algorithm logics offline, each optimized for a candidate tradeoff point, then selects the most appropriate one to run online. We first show how FLAS can be applied to existing algorithms to make them latency-aware. Second, we developed a novel Genetic Programming approach to fully explore FLAS's potential. Extensive evaluations show that FLAS can precisely control latency all the way down to 1s and achieve substantially higher QoE than state-of-the-arts. FLAS can be readily implemented into real streaming platforms, offering a practical solution for live-streaming services.

Index Terms—Live Video Streaming, Mobile Network, Playback Latency, Quality-of-Experience.

1 Introduction

MOBILE video streaming has seen tremendous growth in the past decade and is now a mainstream application in the mobile Internet. Beginning with streaming pre-encoded contents, i.e., on-demand streaming, a new trend in recent years is the streaming of live events, from professionally-authored live contents (e.g., news, concerts, and sports), to user-generated live streams (e.g., personal live shows, game live). This trend is further fueled by the widespread adoption of live-streaming platforms such as YouTube Live [1] and Facebook Live [2].

In addition to the usual quality-of-experience (QoE) metrics such as video quality and playback rebuffering, live video streaming has a unique and important performance criterion – *playback latency*, defined as the time difference between video rendering and actual capturing (note that in this paper, in order to distinguish playback latency from the usual QoE metrics, we don't incorporate latency into the calculation of QoE).

In general, live streaming services require low playback latency (a few seconds at most). However, an important fact is that playback latency and the usual QoE metrics (e.g., quality, rebuffering) are inherently conflicting objectives. For instance, viewers generally prefer to stream high-quality videos which would inevitably incur a longer transmission delay at the mobile radio link. As the transmission delay translates directly into the playback latency of live streaming, the need for high-quality videos inherently conflicts with the live streaming's low latency demand. Therefore, this requires live streaming services to seek performance tradeoffs between the playback latency and QoE.

In practice, different types of live streaming services can have very different latency-QoE tradeoff requirements [3-5]. For example, highly interactive live streams (e.g., live sales, interactive live shows) demand much lower latency than one-way live broadcasts (e.g., news, concerts), but the interactive streams need to tolerate relatively lower video quality than the one-way broadcasts. Therefore, how to achieve *desired* and *optimal* latency-QoE tradeoff performance for different live streaming services is a significant challenge for designing live streaming algorithms.

While many sophisticated live streaming algorithms (e.g., [6-28]) have been proposed in recent years, none of them have addressed the above challenge. Moreover, our investigations revealed that the playback latency achieved by these existing algorithms are far from consistent, but vary over a wide range (e.g., 2s ~ 31s) in changing network conditions. In other words, streaming the same video from the same mobile operator, even in the same location, could result in significantly different latency, depending on the specific network condition experienced. This is clearly undesirable as it is even not possible for the existing algorithms to keep consistent latency, let alone achieve the service-desired latency-QoE tradeoff.

- G. Zhang and J.Y.B. Lee are with the Department of Information Engineering the Chinese University of Hong Kong, Shatin, NT, Hong Kong SAR. E-mail: {ghzhang, yblee}@ie.cuhk.edu.hk
- K. Liu is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Sciences (UCAS), Beijing, China. E-mail: liuke@ict.ac.cn
- H. Hu is with the Department of Electronic and Information Engineering, Hong Kong Polytechnic University, Kowloon, Hong Kong and PolyU Shenzhen Research Institute. E-mail: haibo.hu@polyu.edu.hk
- V. Aggarwal is with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA and Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA. E-mail: vaneet@purdue.edu

To tackle the challenge, we propose a novel framework called Flexible Latency Aware Streaming (FLAS) that not only achieves consistent low latency across a wide range of network conditions, but also can control the latency-QoE tradeoff flexibly. Specifically, we introduce the notion of *state quantizer* (SQ) to quantify the latency-QoE tradeoff under different network conditions into a set of states. FLAS then generates a set of algorithm logics where each member is optimized for a particular state. At runtime, service providers (or viewers) are allowed to specify a target playback latency (e.g., 2s) according to the live-service requirement. With the target latency prescribed, FLAS can periodically select/adjust the operational streaming algorithm based on SQ, so that the actual latency will not deviate from the target while QoE can be maximized.

This work makes three key *contributions*. First, our extensive experiments demonstrated the problem in the current live streaming system – it is fundamentally difficult for existing streaming algorithms to keep consistent low latency, and to achieve desired latency-QoE tradeoffs based on live-service requirements.

Second, we designed FLAS to be a unified framework to make the existing streaming algorithms latency aware. To demonstrate this applicability, we applied FLAS to one state-of-the-art algorithm L2AC [27] and the resultant system is called FLAS-L2AC (note that L2AC was developed upon A3C [29] – a leading-edge deep-reinforcement-learning technique that has been widely used in the design of on-demand streaming algorithms [17,21]). We discovered that although FLAS-L2AC can achieve substantially better performance, its efficacy is still restricted under challenging network conditions, presumably due to the inherent structure of the neural network model adopted [27].

Third, to get rid of the limitation of FLAS-L2AC, we turned to a radically different machine-learning approach – Genetic Programming (GP) [30], which represents candidate solutions in the form of expression trees. Different from deep reinforcement learning using predefined neural network structures, GP does not impose a rigid structure on the expression tree, so one can explore the solution space freely with GP. Based on the insight, we developed FLAS-GP to fully exploit FLAS's potential. FLAS-GP encodes streaming algorithms using expression trees and then executes a novel *latency-aware evolutionary process* to evolve the algorithms for better performance.

Extensive evaluations show that, FLAS-GP 1) can achieve substantially higher QoE with the same or lower playback latency than state-of-the-art streaming algorithms; 2) can precisely control the latency all the way down to 1s; 3) exhibits remarkable spatial and temporal robustness; and 4) consolidates most of the complexities into offline training, leaving a lightweight online implementation that can be easily deployed on real streaming platforms.

The rest of the paper is organized as follows: Section 2 reviews the background and related work; Section 3 reveals the problems of existing streaming algorithms; Section 4 presents the FLAS framework and evaluates the efficacy of applying FLAS to L2AC; Section 5 presents a novel Genetic Programming approach to fully explore FLAS's potential; Section 6 evaluates and compares the performance of FLAS against current state-of-the-arts, and Section 7 summarizes the study and outlines some future work.

2 Background and Related Work

Adaptive video streaming is the primary tool service providers use to compensate for the inevitable bandwidth fluctuation in mobile and fixed networks. Given the widespread use of video streaming, researchers have developed many novel adaptive streaming algorithms for on-demand streaming in recent years. The basic principle is to design algorithms to dynamically select the future video bitrate in the light of past measurements such as throughput and buffer occupancy. Existing adaptive streaming algorithms can be classified based on their measured metrics, e.g., bandwidth-based [7-8], buffer-based [9-10], and hybrid-bandwidth-buffer-based [11-21] approaches, or classified based on the technique employed in optimizing the adaptation algorithm, e.g., heuristics [7-12], PID-controller [13], data-analytic [14-16], machine learning [17-21].

In live streaming, however, playback latency is the primary performance metric. While the above adaptive streaming algorithms worked well for on-demand streaming, they often exhibited latency too high to be suited for live streaming services. Therefore, a number of researchers have begun developing new adaptation algorithms specifically for live streaming services. One of the problems in live streaming is that the data buffering process can directly increase playback latency. This motivates designs to control the amount of buffered video data in the streaming pipeline to reduce the latency. For example, Cicco *et al.* [22] proposed a client-side algorithm employing PID feedback control to track a target buffer occupancy by adapting the video bitrate, thereby maintaining low latency while preventing playback rebuffering. Similarly, Wang *et al.* [23] proposed a PID-based adaptation algorithm to control the buffer occupancy at the streaming server side. Xie *et al.* [24] proposed DTBB to select video bitrate depending on a buffer threshold that is tuned dynamically according to measured network throughput.

The second problem in live video streaming is latency *accumulation* where the primary source of latency is playback rebuffering. Specifically, during rebuffering where the player runs out of video data, video playback will be suspended until sufficient data are downloaded to resume the playback. The live event, on the other hand, continues on and thus the gap between the video playback and the actual capturing will be widened by the rebuffering duration. Given that video data are played in sequence, latency once introduced thus cannot be reduced, resulting in incremental latency throughout the streaming session whenever rebuffering occurs.

Two common methods to solve this problem are video data skipping and playback rate regulating. The former one is to skip the download/playback of the late-arriving video segments while the later one is to accelerate the video playback. Both of them can make the video player catch up with the live event.

TABLE 1 Evaluation Settings

Streaming parameters	Values
Bitrate profile	{0.2, 0.4, 0.8, 1.2, 2.2, 3.3, 5.0, 6.5, 8.6} Mbps [37]
Segment duration	2s
Frame rate	25 fps
Live event duration	3600s
Client buffer size	60s
Initial video bitrate	0.2 Mbps
Throughput trace	60 days for training and 60 days for testing

For instance, Miller *et al.* [25] proposed LOLYPOP that executes data skipping once the playback latency is larger than a pre-defined skipping threshold. Lim *et al.* [26] proposed LoL that turns up the video playback rate to achieve low latency. Zhao *et al.* [27] developed L2AC that incorporates both of the two methods, i.e., determining the skipping threshold and the playback rate simultaneously through neural networks trained by deep-reinforcement-learning.

Although the above algorithms were designed with reducing latency in mind, this is still far from enough in practice, as different types of live services can have very different latency and QoE requirements. Recently, Zhang *et. al* [28] proposed LAPAS that allows users to specify a target playback latency according to live-service requirements, and then LAPAS tracks the target through running a streaming-parameter optimization (i.e., tuning the streaming parameters based on the network condition). However, LAPAS suffers from two fundamental limitations: First, as LAPAS's adaptation logic is a fixed heuristic restricted by human intuitions, it fails to achieve optimal performance across a broad set of network conditions; Second, the computational overhead of the streaming-parameter optimization is very large as it utilizes brute-force search, which hinders the large-scale deployment of LAPAS in real streaming platforms.

By contrast, the FLAS framework developed in this study offers three superiorities over the existing approaches. 1) In addition to achieving consistent low latency across a wide range of network conditions, FLAS can enable flexible latency-QoE tradeoff control; 2) Instead of relying on pre-programmed models from human intuitions, FLAS is designed as a unified framework, which not only can be applied to optimizing the existing algorithms, but also guides the design of new latency-aware approaches. For instance, we apply FLAS to L2AC (the state-of-the-art algorithm) in Section 4, and explore a novel FLAS-based Genetic Programming approach in Section 5. This feature enables FLAS to incorporate any advanced techniques to fully liberate the performance potential on its own; 3) The two-phase design of FLAS avoids difficulties in real platform deployment, bringing a completely practical solution to current live streaming services.

3 Evaluation of Existing Algorithms

In this section, we evaluate the performance of five leading-edge adaptive live streaming algorithms and then demonstrate their limitations.

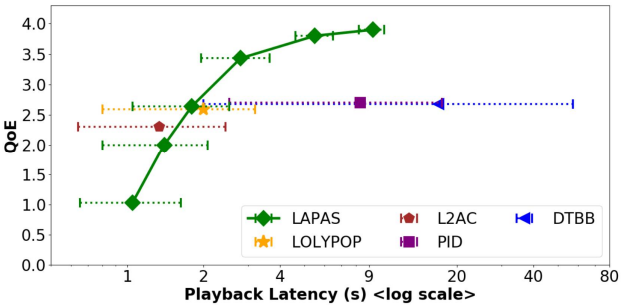

Fig. 1. Comparison of mean QoE and playback latency (error bars span streaming sessions with top/bottom 5% latency).

3.1 Experiment Setup

To evaluate the performance of streaming algorithms in realistic network settings, we employed trace-driven simulations where the simulator executes the streaming algorithms over simulated network conditions reproduced by TCP throughput trace data.

Specifically, a content provider captured a live video stream through a camera, and then uploaded it to a streaming server where the video would be encoded into multiple bitrate versions following Apple's recommended bitrate profile [37]. Common Media Application Format (CMAF) [33] was adopted into DASH [32] so that the streaming server can deliver the video data on a frame-by-frame basis to minimize segmentation delay. A video player began playback after the first video frame was downloaded. The initial video bitrate was set to the lowest level (i.e., 0.2 Mbps). The network condition was emulated by replaying TCP throughput trace data captured from multiple production mobile networks, including 3G, 4G, and Wi-Fi [36,39,40]. The rest of the parameters are summarized in Table 1.

Two primary performance metrics were adopted: 1) *mean playback latency* is defined as the average playback latency experienced in each streaming session, and 2) *QoE* is calculated from the QoE function proposed by Yin *et al.* [11] combined with a penalty for video data skipping [27] (the component weight also follows [11] and [27]):

$$Q = \frac{1}{K}\left(\sum_{k=0}^{K-1} r_k - \sum_{k=1}^{K-1}\left|r_k - r_{k-1}\right| - 3.0\times Z - 3.0\times Z' - 0.2\times G\right) \quad (1)$$

where Z is the rebuffering duration, Z' is the startup delay, r_k is the bitrate selected for segment k in Mbps, G is the skipped video duration, K is the total number of segments in one streaming session. We will further consider other QoE metrics in Section 6.2.

We evaluated five state-of-the-art adaptive live streaming algorithms: LOLYPOP [25], LAPAS [28], L2AC [27], PID [23], DTBB [24]. We referred to the source codes provided by Zhao *et al.* [40] to train the neural networks in L2AC. A total of 60 days throughput trace data (~50,000 streaming sessions) were used for the training and another unseen 60 days traces were applied to evaluating the performance of the five streaming algorithms.

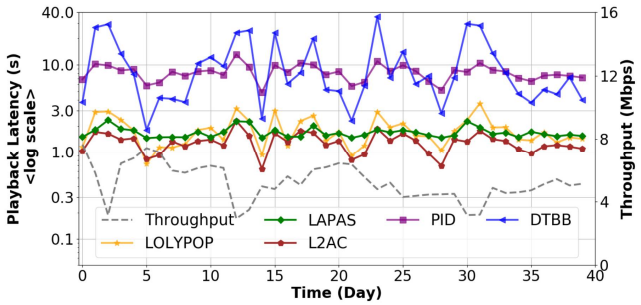

Fig. 2. Comparison of daily latency over a period of 40 days.

3.2 Results and Discussions

Observation 1. Fig. 1 compares the mean QoE and playback latency of the five streaming algorithms in all tested streaming sessions. Most streaming algorithms (except LAPAS) *do not* have latency awareness, so each of them achieves only one specific point of tradeoff between QoE and latency. This is a significant limitation in practice as different types of live services or video contents can have very different latency/QoE requirements. By comparison, LAPAS obtains a continuous tradeoff frontier between QoE and playback latency, as LAPAS supports the setup of target latency and offers a streaming-parameter optimization to keep tracking the target. Interested readers can refer to [28] for more details.

However, the performance of LAPAS is much worse than L2AC. For example, there is a 15.2% QoE gap between them under the playback latency 1.4s. We argue that this is because LAPAS's adaptation logic is a fixed heuristic preprogrammed by humans, which is inevitably restricted by human intuitions, thus limiting the performance. In comparison, L2AC employs a deep learning technology (i.e., A3C [29]) and learns a better adaptation logic based on its past experience. Nonetheless, as we mentioned, L2AC is not a latency-aware algorithm (only have a single tradeoff point), so it is incapable to achieve latency-QoE tradeoffs based on requirements.

Observation 2. In Fig. 1, we found that there is a large variation in latency across different algorithms, ranging from 1.4s (L2AC) to 19s (DTBB), and their error bars indicate that the latency achieved in different streaming sessions vary over a wide range as well. Hence we plotted Fig. 2 to show latency variations over a period of 40 days by using a 40-day TCP throughput trace. To appreciate the variations in network conditions, we plotted the daily mean TCP throughput in Fig. 1, which fluctuates significantly from a low of 3.3 Mbps to a high of 7.8 Mbps.

As expected, the daily mean latency of most streaming algorithms (except LAPAS) fluctuate substantially with the changing network conditions, e.g., the latency of DTBB ranges from 2s to 31s. This is clearly undesirable as they even cannot keep consistent low latency, let alone achieve the service-desired latency-QoE tradeoff. By contrast, LAPAS exhibits more consistent latency over the 40 days (we set a 2s target latency). This is because through analyzing the throughput traces in past streaming sessions, LAPAS's streaming-parameter optimization can periodically tune the value of the streaming parameters to adapt to the changing network conditions.

However, this is at the expense of extremely high computational complexity, as the parameter optimization is built upon brute-force search. Specifically, based on our test, one-day optimization even costs ~1320 CPU hours (for 4-parameter tuning [28]). To keep playback latency consistent, LAPAS requires the streaming server to run it on a daily basis. This undoubtedly occupies a large amount of computational resource, especially when the streaming client scales, and thus hinders the large-scale deployment of LAPAS on real streaming platforms.

Two insights. 1) Since most existing algorithms adopt fixed adaptation logics with immutable streaming parameters (so-called one-size-fits-all mode), they don't have latency awareness. In contrast, LAPAS is latency-aware as it is able to tune a specific set of streaming parameters accordingly for different latency requirements and network conditions; 2) Although LAPAS is a feasible solution, only tuning the parameters but without adapting the algorithmic logic may still be insufficient to achieve optimal performance. Besides, LAPAS also suffers from deployment issues in practice.

Our Approach. To tackle the limitation of the existing approaches, we propose to optimize/train different streaming algorithm logics specifically for different latency requirements and network conditions (instead of only tuning streaming parameters) where each logic covers a subset of the target environment and together provide full coverage. Equipped with these candidate algorithm logics, the system can then conduct online algorithm selections and adjustments to keep the most appropriate streaming algorithm always in operation, so that a full spectrum of latency-QoE tradeoffs can be offered under a broad set of network conditions.

4 Flexible Latency Aware Streaming

Inspired by the insights from Section 3, we develop a unified framework called Flexible Latency Aware Streaming (FLAS). FLAS is built upon two phases where most of the complexities are consolidated into *distributed offline training phase* to train a set of algorithm logics, and a lightweight strategy is incorporated in *online algorithm selection phase*. The system architecture of FLAS is depicted in Fig. 3.

While streaming a live video event, the network condition may change significantly, so the initial-selected algorithm may no longer be optimal later on. We thus propose to divide one streaming session into multiple sub-sessions (henceforth called "*epoch*") where each has a fixed video duration (e.g., 300s), so the system can execute algorithmic *re-selection* in the interval of each epoch (within the streaming session) to adapt to the changing network conditions. This is illustrated in Fig. 3 and we will introduce more details later.

In this section, we first introduce the design of the FLAS framework and then demonstrate a use case of FLAS by applying it to optimizing an existing streaming algorithm, L2AC [27].

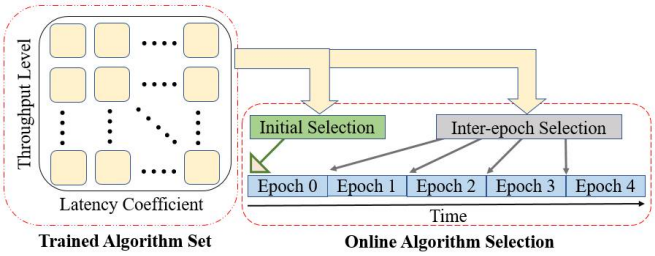

Fig. 3. The system architecture of FLAS.

4.1 Distributed Offline Training

To quantify the latency-QoE tradeoff under different network conditions, we define a two-dimensional *state quantizer* (SQ), denoted by

$$\vec{\psi} = \langle \omega, \varpi \rangle \tag{2}$$

where the first dimension ω is called *latency coefficient* and the second dimension ϖ is named as *throughput level*.

Latency Coefficient ω. The function is to extract the relationship between playback latency and QoE. Specifically, let μ_k be the time elapsed since the beginning of the live event at the time of requesting segment k. The current playback time point, denoted by l_k, is known to the video player so that the playback latency α_k can be computed from

$$\alpha_k = \mu_k - l_k \tag{3}$$

The mean playback latency over all video segments in one epoch is then given by

$$\beta = \frac{1}{K}\sum_{k=0}^{K-1}\alpha_k \tag{4}$$

where K is the total number of video segments in an epoch.

The mean latency β is then combined with the desired QoE function (e.g., (1)) to form the objective function used in the offline training:

$$U = Q - \omega \times \beta \tag{5}$$

where Q is the QoE function and ω is the *latency coefficient*. The offline training will maximize the objective function U where the playback latency β and QoE Q are conflicting metrics with each other, so the latency coefficient ω can be tuned to balance the tradeoff between them. For instance, a larger value of ω will result in an algorithm trained with lower latency but worse QoE.

Based on this principle, we define M values of ω, i.e., $\{\omega_p | p=0,1,\ldots,M\text{-}1\}$, to generate M objective functions:

$$U_p = Q - \omega_p \times \beta,\ p=0,1,...,M-1 \tag{6}$$

which can quantify M candidate latency-QoE tradeoff points.

Throughput level ϖ. Since the playback latency varies across different network conditions (c.f. Observation 2 in Section 3.2), we introduce throughput level to differentiate the network conditions where throughput level is defined as the mean throughput during streaming video sessions (we also tested some other metrics, e.g., throughput variation and network type, but they perform worse). While throughput level can be calculated directly in offline as the throughput trace data is given, it cannot be known before streaming the actual video in online streaming. Therefore, to keep the calculation process consistent, we propose the following way to *estimate* throughput level.

Specifically, the throughput of a new video epoch j can be estimated from the mean throughput of downloading the last m segments in the last epoch j-1:

$$V_j = \frac{1}{m}\sum_{k=0}^{m-1}\frac{s_{j-1,k}}{d_{j-1,k}} \tag{7}$$

where $s_{j-1,k}$, and $d_{j-1,k}$ are the size and download time of segment k in epoch j-1. We then apply a linear quantization policy to map the measured throughput V_j to a discrete throughput W_j:

$$W_j = \min\left(\left\lfloor \frac{V_j}{\Delta} \right\rfloor, N-1\right) \tag{8}$$

where Δ is the quantization step size and N is the maximum value of the discrete throughput. The next step is to segregate the throughput trace data of all epochs S_j, j=0,1,…,J, into N network classes through *throughput level* ϖ_q:

$$C_q = \left\{ S_j \middle| W_j = \varpi_q, \forall j \right\},\ \ q = 0,1,\ldots,N-1 \tag{9}$$

where each throughput trace class will emulate a particular network condition in the offline training.

Training. With the two-dimensional SQ, i.e., M latency coefficients and N throughput levels, FLAS can quantify a total of M×N latency-QoE tradeoff states:

$$\prod = \left\{ \left(U_p, C_q\right) \middle| p = 0,1,...,M-1,\ q = 0,1,...,N-1 \right\} \tag{10}$$

where U_p is the objective functions (to be maximized) with latency coefficient ω_p (defined (6)) and C_p is the throughput trace data with throughput level ϖ_q (defined in (9)). For each state, FLAS runs a separate training process, denoted by function $T_x(\cdot)$, to train a specialized adaptation algorithm for this state:

$$A_{p,q} = T_x(U_p, C_q),\ p = 0,1,...,M-1,\ q = 0,1,...,N-1 \tag{11}$$

where $A_{p,q}$ is the trained algorithm set including M×N algorithms. Fig. 3 illustrates the trained algorithm set where each square represents an adaptation algorithm trained for one state. During the training, FLAS also records the resultant QoE, denoted by $Q_{p,q}$, and mean playback latency, denoted by $\beta_{p,q}$, achieved by the algorithm of each state, which will then be utilized in the online algorithm selection phase (c.f. Section 4.2).

Importantly, FLAS is designed as a *general* framework that is able to operate upon any underlying adaptation algorithms. For instance, when applying FLAS to L2AC (i.e., FLAS-L2AC), the training function (i.e., $T_x(.)$ in (11)) is deep reinforcement learning A3C [29], and the resultant algorithm set (i.e., $A_{p,q}$ in (11)) includes M×N neural networks (we will evaluate the performance of FLAS-L2AC in Section 4.3).

TABLE 2 System Configurations of FLAS

System parameters	Values
Epoch duration	300s
Latency coefficient	A total of 18 coefficients ranging from 0.01 to 1.8
Throughput level	A total of 10 levels with quantization step 1 Mbps
PI controller	K_p=0.5, K_i=0.05 [35]

4.2 Online Algorithm Selection

The algorithm set trained in the offline training will be downloaded to the video player (e.g., through DASH metadata [32]) for online streaming. To cater to different playback latency requirements, FLAS supports runtime configuration of target playback latency (e.g., 2s), denoted by λ, which can be input as a video player option. With the target latency λ prescribed, the system's goal is to prevent the actual latency from deviating from the target while maximizing QoE.

Initial Selection. FLAS client first selects the most appropriate adaptation algorithm running at the beginning of a streaming session, i.e., in *epoch 0*. Since all the trained algorithms are labeled with different states, the state quantizer (SQ) can be used to assist in the algorithm selection process.

The streaming client begins a live streaming session by prefetching m video segments with a pre-configured fixed bitrate where the total prefetching video duration equals to the target latency λ. FLAS client then measures the average throughput in downloading these m video segments:

$$V_0 = \frac{1}{m}\sum_{k=0}^{m-1}\frac{s_{0,k}}{d_{0,k}} \tag{12}$$

where $s_{0,k}$, and $d_{0,k}$ are the size and download time of the segment k in epoch 0. The average throughput V_0 will then be utilized to estimate throughput level:

$$\varpi_{q^*} = \min\left(\left\lfloor \frac{V_0}{\Delta} \right\rfloor, N-1\right) \tag{13}$$

where Δ is the quantization step size and N is the maximum value of throughput levels.

After obtaining the throughput level, the next step is to determine latency coefficient. Specifically, FLAS will select the operational algorithm among the algorithms trained with throughput level ϖ_{q^*} and M different latency coefficients $\{\omega_p | p=0,1,\ldots,M\text{-}1\}$. The decision criteria is to find which algorithm can achieve the highest QoE while its playback latency does not exceed the latency target λ:

$$\max_p Q_{p,q^*} \quad \text{s.t. } \beta_{p,q^*} \le \lambda,\ p=0,1,\ldots,M-1 \tag{14}$$

where Q_{p,q^*} and β_{p,q^*} are the QoE and playback latency achieved by the algorithm trained with latency coefficient ω_p and throughput level ϖ_{q^*} (these data are recorded in the offline training, c.f. Section 4.1). Finally, the video player will apply the matching adaptation algorithm to epoch 0.

Inter-epoch Selection. During streaming the subsequent epochs (e.g., epoch 1~epoch 4 in Fig. 3), network conditions can change significantly, so the FLAS client will adjust the operational adaptation algorithm at the start of each epoch to keep the actual latency from deviating from the target λ.

Specifically, at the beginning of epoch j (j>0), FLAS activates a PI feedback controller:

$$u_{j-1} = K_p e_{j-1} + K_i \sum_{x=0}^{j-1} e_x \tag{15}$$

where K_p and K_i are tuning parameters representing the proportional gain and integral gain of the PI controller [35], and

$$e_{j-1} = \xi_{j-1} - \lambda \tag{16}$$

is the deviation of the actual mean latency in epoch j-1, denoted by ξ_{j-1}, from the target latency λ.

To compensate for the latency deviation incurred in epoch j-1, i.e. (16), the PI controller will adjust the target latency of epoch j to λ_j by:

$$\lambda_j = \lambda_{j-1} - u_{j-1} \tag{17}$$

where λ_{j-1} is the target latency adopted in epoch j-1 and u_{j-1} is the output of the PI feedback controller, i.e., (15). An intuitive example to illustrate the principle of this feedback mechanism is that, if the actual latency is higher than the target latency in the current epoch, one can set a lower target in the next epoch to shorten the actual latency so that the deviation can be reduced.

With the new target latency λ_j, the next step is to determine the operational algorithm for use in epoch j through SQ. First, throughput level is estimated by the mean throughput in downloading the last m segments in epoch j-1:

$$V_j = \frac{1}{m}\sum_{k=0}^{m-1}\frac{s_{j-1,k}}{d_{j-1,k}} \tag{18}$$

where $s_{j-1,k}$, and $d_{j-1,k}$ are the size and download time of segment k in epoch j-1. The mean throughput V_j is then mapped to a discrete throughput level by

$$\varpi_{q^*} = \min\left(\left\lfloor \frac{V_j}{\Delta} \right\rfloor, N-1\right) \tag{19}$$

where Δ is the quantization step size and N is the maximum value of throughput level.

Latency coefficient is determined by

$$\max_p Q_{p,q^*} \quad \text{s.t. } \beta_{p,q^*} \le \lambda_j,\ p=0,1,\ldots,M-1 \tag{20}$$

where the underlying principle is identical with (14), except λ_j replacing λ, which is adjusted by the PI feedback controller in (15)~(17). After determining SQ, the video player will apply the matching adaptation algorithm to epoch j.

4.3 Performance Evaluation

FLAS is a general framework that can optimize existing adaptive streaming algorithms to make them latency aware. To evaluate this applicability, we applied FLAS to optimizing the state-of-the-art algorithm L2AC [27]. The resultant system is named as *FLAS-L2AC*.

TABLE 3 QoE Performance Across Throughput Levels (Target Latency=2s)

Algorithm	Throughput Level				
	0~1	2~3	4~5	6~7	8~9
FLAS-L2AC	-1.11	1.72	3.06	4.83	6.97
LAPAS	0.21	1.79	2.79	4.07	5.20

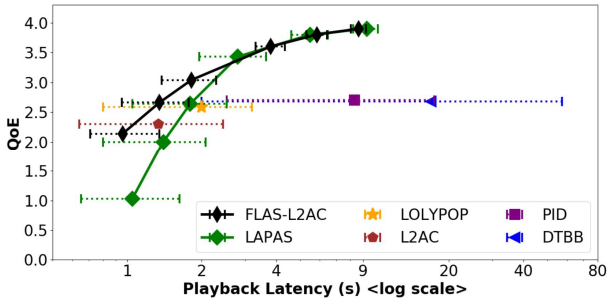

Fig. 4. Comparison of QoE and playback latency between FLAS-L2AC and the existing algorithms (error bars span streaming sessions with top/bottom 5% latency).

We conducted the trace-driven simulation as described in Section 3 where Table 1 summarizes the simulation settings. For the configuration of FLAS, we adopted 14 latency coefficients (defined in (6)) ranging from 0.01 to 1.8, and 10 throughput levels (defined in (8)) with 1 Mbps quantization step size. Unless stated otherwise, the live events last for 3600s and the epoch duration is set to 300s. The rest of the parameters are summarized in Table 2.

Fig. 4 compares the latency-QoE tradeoff of FLAS-L2AC to the existing streaming algorithms. We observed that FLAS configures L2AC to offer a continuous tradeoff frontier. Remarkably, FLAS improves the QoE performance of L2AC by 15.7% under the playback latency ~1.4s. This strongly suggests that FLAS and L2AC can cooperate effectually to train more specialized neural networks that better match the operating environments. Compared to LAPAS, FLAS-L2AC achieves much higher QoE in lower latency targets (≤3s), while performing similarly in higher latency targets (>3s).

Fig. 5 plots the daily mean latency over a period of 40 days. Same as LAPAS, we set the target latency to 2s for FLAS-L2AC. We observed that among all the algorithms, only FLAS-L2AC and LAPAS exhibit consistent latency and track the latency target closely over the 40 days. As opposed to LAPAS requiring daily optimization (discussed in Section 3), FLAS-L2AC does not need to repeat the training process at all, as the latency variations largely due to network condition changes have already been addressed by FLAS. Moreover, FLAS's training process carries most of the system complexity and once it is completed no need to be re-executed, so that the implementation and deployment for FLAS are much simpler than LAPAS.

Limitations of FLAS-L2AC. Fig. 6 compares the daily mean QoE achieved by FLAS-L2AC and LAPAS under 2s target latency over the 40 days. We also plotted the daily mean TCP throughput to appreciate the variation of network conditions.

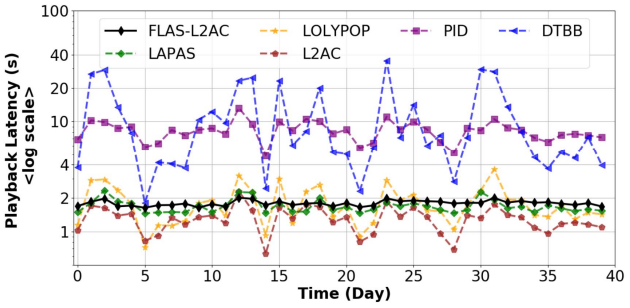

Fig. 5. Comparison of daily playback latency between FLAS-L2AC and the existing algorithms over a period of 40 days.

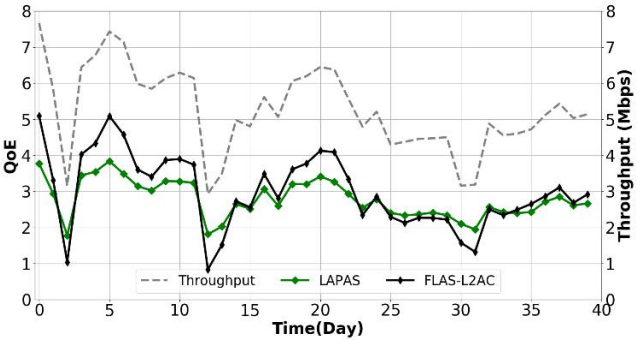

Fig. 6. Comparison of daily QoE performance between FLAS-L2AC and LAPAS over a period of 40 days.

We can see that the QoE tracks the throughput closely, as the latter directly impacts the mean video bitrate delivered, which is the primary factor affecting QoE. Moreover, a more interesting observation is that, compared to LAPAS, FLAS-L2AC only achieves better QoE performance in 28 of the 40 days but performs worse in the rest 12 days. Comparing the daily throughput, it appears that FLAS-L2AC becomes *less effective* in networks with lower throughput levels.

To this end, we further studied the QoE performance across different throughput levels. We divided all streaming sessions into 10 throughput levels, with level l=0,1,…,8 collecting sessions with average throughput within (l, l+1] Mbps, plus level 9 with average throughput ≥9Mbps, and then summarized the respective QoE performance of FLAS-L2AC and LAPAS in Table 3. The results verify our conjecture that FLAS-L2AC performs much worse in lower throughput levels, especially at the lowest two throughput levels (i.e., 0~1). We also found similar results in other target latency options.

One of the challenges in using machine-learning approaches to solve problems is that the resultant solutions (e.g., neural network) are often opaque and difficult to analyze so that the insights into their performance cannot be easily obtained. To shed light on the results in Table 3, we attempted to tackle the challenge by analyzing the streaming algorithm's bitrate adaptation behavior. Specifically, by fixing other less critical parameters, e.g., set buffer occupancy to 2s and the last segment bitrate to 200kbps, we can plot the bitrate decision (y-axis) versus measured throughput (x-axis) for the adaptation logics.

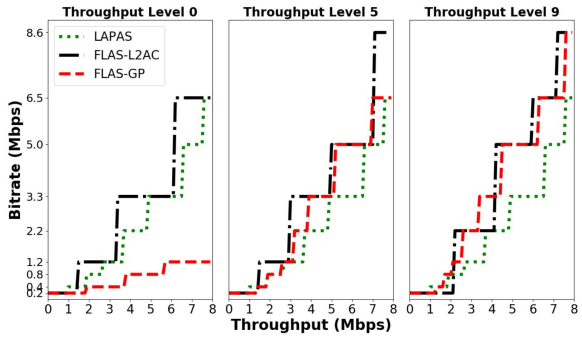

Fig. 7. Comparison of bitrate adaptation behavior in low (0), medium (5), and high (9) throughput levels (target latency = 2s).

We plotted the results in Fig. 7 for low (0), medium (5), and high (9) throughput levels where the calibration values of the y-axis indicate the available video bitrate versions. For FLAS-L2AC, the results reveal one of its behavior – its bitrate adaptation logic is relatively aggressive. While this may work well in high throughput levels, it would cause disastrous consequences in low throughput levels that typically have substantial throughput fluctuations. Therefore, this clearly explains why FLAS-L2AC cannot perform equally well across different throughput levels. Another issue of FLAS-L2AC is the abrupt changes of the bitrate decision boundary. For example, the bitrate changes sharply from 0.2 Mbps to 1.2 Mbps (from 0.2 Mbps to 2.2 Mbps in level 9) despite the availability of two intermediate bitrate choices (0.4 Mbps and 0.8 Mbps). We conjecture that in spite of using the state-of-the-art deep reinforcement learning A3C, the resultant neural network structure may still not be sufficiently flexible to explore the complete solution space of the bitrate adaptation.

In comparison, LAPAS exhibits relatively conservative behavior at throughput level 0, which is more reasonable. However, limited by its fixed heuristic logic, the conservatism in bitrate selection cannot be appropriately altered in higher throughput levels, so this inevitably results in suboptimal performance.

GP Scheme. To further explore FLAS's potential, we will turn to a new approach in Section 5 – Genetic Programming (GP). The preliminary results of FLAS-GP (i.e., applying FLAS to GP) are plotted in Fig. 7 which presents a far more reasonable bitrate adaptation behavior. We can observe that, as the measured throughput raises, the selected bitrate of FLAS-GP gradually increases without any abrupt changes (unlike FLAS-L2AC). Specifically, at level 0 it is clear that FLAS-GP intentionally selects bitrates much lower than the measured throughput, as the network condition is judged to be poor and high measured throughput would be treated as exceptions that are unlikely to last. Thus not raising the bitrate too far would effectively prevent rebuffering in the future. By comparison, at level 5, FLAS-GP becomes more moderate and balanced in its bitrate selection. Finally, at level 9, FLAS-GP becomes more aggressive, even occasionally selecting bitrates higher than the measured throughput. Intuitively, at throughput level 9, the low measured throughput is likely short-term so maintaining high video bitrates can prevent unnecessary QoE degradations. We will introduce the design of FLAS-GP in the next section.

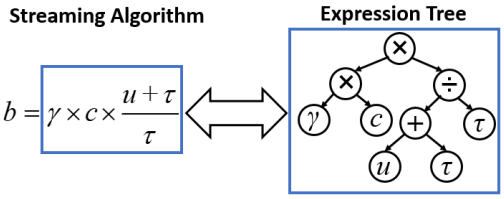

Fig. 8. Illustration of the transform between adaptation algorithm and GP expression tree (c, u are estimated throughput and buffer occupancy respectively; γ and τ are numeric constants; b is video bitrate).

5 Flexible Latency Aware via GP

Genetic Programming (GP) [30] is inspired by the process of natural selection where a population evolves itself to adapt to the changing environment through crossover, mutation, and reproduction.

Why GP? GP encodes candidate solutions in the form of expression trees and does not impose a rigid structure on them, so GP-based schemes can be free to explore the solution space. This is totally different from deep-reinforcement learning which holds a predefined fixed neural network structure and only the neuron weights can be tuned. Therefore, using GP can potentially resolve the limitations of FLAS-L2AC.

In this section, we investigate the GP approach for FLAS where adaptation algorithms are encoded with expression trees. In addition, a new latency-aware evolutionary process is developed to make GP more suitable for evolving streaming algorithms in the scenario of live video streaming. It is worth noting that other machine learning or heuristic techniques can be operated by FLAS in a similar manner, and this could be a fruitful direction for future work.

5.1 Adaptation Algorithm and Expression Tree

GP encodes candidate solutions using *expression trees* [30] which are particularly suitable for representing streaming algorithms. In fact, many existing adaptive streaming algorithms can be mapped to relatively simple expression trees. Fig. 8 shows an example in on-demand streaming: the right-side expression tree is the equivalent of the left-side expression, which is a hybrid-throughput-buffer-based adaptation algorithm proposed by Liu *et al.* [14]. The algorithm determines the bitrate b according to the estimated throughput c and the buffer occupancy u. In the following, we will apply expression trees to encoding *playback/bitrate adaptation algorithms* for live video streaming.

Playback Adaptation. A fundamental problem in live streaming is the *accumulation* of playback latency where the primary source of latency is playback rebuffering. Specifically, rebuffering occurs when the video player runs out of video data and thus has to suspend video playback until more video data are received. The live event, on the other hand, continues on and thus the time gap between the video playback and the actual capturing will be widened by the rebuffering duration. Worst still, as subsequent video data are played back in sequence, the widened gap will be eventually accumulated into the playback latency for the rest of the streaming session. In fact, whenever a rebuffering event occurs, the playback latency will be in-

creased by the rebuffering duration. This is clearly undesirable in live video streaming, as the latency would keep increasing throughout the whole streaming session.

To the best of our knowledge, there are two effective methods to address this problem. The first one is video data skipping [25], i.e., by skipping the download/playback of the late-arriving video segments, the video player can then catch up with the live event. However, this also introduces playback glitch as a tradeoff thereby resulting in QoE degradation [27]. By comparison, the second one, i.e., regulating the playback rate [26,28], has much fewer impacts on QoE. The idea is to increase the video playback framerate slightly (e.g., within 5%) to catch up with the live event. Such a slight change to the playback rate is much less perceivable to viewers and thus can prevent the QoE degradation [31].

Therefore, instead of using video data skipping, we employ playback rate regulation to control the playback latency in our following design. Fig. 9 illustrates the relationship between playback latency, rebuffering, and playback rate adaptation. The x-axis is wall-clock time while the y-axis is the playback point in the video stream. Assuming the live streaming session starts at time point zero then the live event's timeline is a 45-degree line passing through the origin. A video player streaming live events will first buffer video data up to the target latency (2s in this example) before commencing playback, thereby resulting in an initial latency of 2s. When a rebuffering event occurs at time t_1, the client suspends video playback for 1s before resuming it at time t_1+1. Due to the rebuffering event, the playback latency is then increased to 3s, thus exceeding the latency target of 2s. To reduce playback latency, the player increases playback rate until the target is reached, after which it reverts back to normal playback rate.

To control the playback rate automatically, we explore the use of Genetic Programming (GP) to evolve playback adaptation algorithms. GP encodes candidate solutions using *expression trees* (see Fig. 8) which comprise two types of components: *operands* – leaf nodes, and *operators* – non-leaf nodes. The choices of operands and operators determine GP's search space. For the playback rate adaptation, GP captures network and streaming states as inputs in an expression tree via *variable operands*. We define a variable operand set $\Re$ with four input variables:

$$\Re=\{\delta, z, b, \alpha\} \tag{21}$$

where δ is the average TCP throughput in downloading the past x (e.g., x=5) video segments; z is the current buffer occupancy; b is the bitrate of the previous video segment; and α is the playback latency. The first three variables are commonly employed in on-demand adaptive streaming algorithms while the last one is specific to live video streaming.

In addition to input variables, one also needs numeric constants for constructing algorithms. These are introduced into GP expression tree via constant numeric operands, defined by an operand set $\Im$ that comprises numeric constants randomly generated over a given range D:

$$\Im=\{x\in\mathbb{R}, -D<x<D\} \tag{22}$$

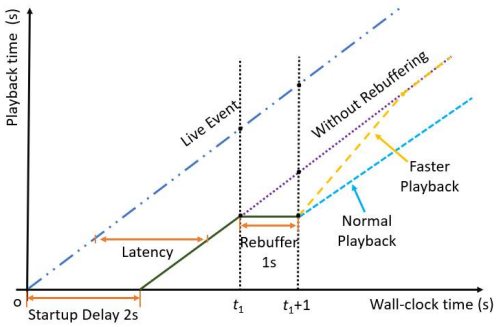

Fig. 9. Illustration of using adaptive playback to control playback latency.

As opposed to operands, GP operators are non-leaf nodes of an expression tree. An operator node performs a specific operation on its child nodes to produce a result which is then served as input to its parent operator node. We employ a set of four arithmetic functions in the operators set $\aleph$:

$$\aleph=\{+,-,\times,\div\} \tag{23}$$

The above operands and operators are specially chosen such that the resultant expression tree can be presented in the form of a mathematical equation that can be simply implemented into video players. The output of the expression tree is playback rate multiplier, denoted by θ. Specifically, θ=1 means normal playback rate and θ>1 (θ<1) speeds up (slows down) playback rate by a factor of θ. Note that slowing down the playback rate serves the purpose to assist in avoiding playback rebuffering by buffering up more data in the case of the actual latency lower than the target.

In practice, one wants to make the playback rate change imperceptible so that it does not degrade the user experience. Previous work [28,31] found that, for both video and audio, playback rate changes within ±5% are imperceivable to most viewers. To verify this, we tested a wider range of playback rate changes (within ±50%) via dash.js video player [41] and our findings were consistent with the previous work. In fact, we found it is difficult for viewers to notice even with a ±15% playback rate change but conservatively we adopted ±5% maximum change limit in this work. Interested readers can visit dash.js reference player [41] to experience it. Finally, we limit the playback rate multiplier computed by the expression tree to this range:

$$\rho=\begin{cases}\min(\theta, 1+\kappa_{\max}), & \theta>1\\ \theta, & \theta=1\\ \max(\theta, 1-\kappa_{\max}), & \theta<1\end{cases} \tag{24}$$

where $\kappa_{\max}$ = 5% is the maximum playback rate change.

Bitrate Adaptation. For evolving bitrate adaptation algorithms, we define a set Φ that includes five domain-specific inputs as *variable operands*:

$$\Phi=\{\delta, z, b, \omega, \rho\} \tag{25}$$

Comparing (25) to (21), an additional operand ρ – the playback rate calculated from (24) is added. This means that the

bitrate adaptation algorithm has knowledge of the playback rate chosen to enable the two adaptation algorithms to evolve jointly (c.f. Section 5.2).

The definitions of numeric constants operands and operators in bitrate adaptation are identical with that in (22) and (23) respectively. Same as the playback adaptation algorithm, the choices of operands and operators in bitrate adaptation also enable the expression trees to be converted into mathematical equations that can be easily implemented into the video player.

The output of the expression tree here is the video bitrate, denoted by r, of the next video segment, but as r is a real number while bitrate choices are discrete, it needs to be mapped to the closest available bitrate version by

$$h=\arg\min_{h}\left|r_h - r\right| \tag{26}$$

where r_h, h=0,1,…,H-1, are the available bitrate versions.

5.2 Latency-aware Evolutionary Process

The two types of algorithms (i.e., playback adaptation and bitrate adaptation) defined in Section 5.1 are not independent but should work together to optimize QoE and playback latency in live video streaming. On one hand, given the different functions performed by the two types of algorithms, they should be evolved in separate GP populations. On the other hand, system performance is a result of running them simultaneously so one also needs a way to evolve them jointly. To this end, we drew on the methodology of cooperative coevolution [34] and developed a new latency-aware evolutionary process to co-evolve the two types of algorithms in live video streaming.

Populations. The evolutionary process begins with two separate initial populations, one for playback rate adaptation and the other for bitrate adaptation, each containing γ (e.g., γ=800) randomly-generated individuals (i.e., expression trees). We adopted the method proposed by Koza *et al.* [30] to generate the initial populations.

Let $\mathbf{I}_{\tau,g}$ ($\mathbf{I}_{\pi,g}$) and $I_{\tau,g,k}\in\mathbf{I}_{\tau,g}$ ($I_{\pi,g,k}\in\mathbf{I}_{\pi,g}$) be the population set and individual k, k=0,1,…,K–1, in the population in generation g, g=0,1,…,G-1, for playback (bitrate) adaptation algorithms respectively. We link each pair of individuals ($I_{\tau,g,k}$ and $I_{\pi,g,k}$) from the two populations ($\mathbf{I}_{\tau,g}$ and $\mathbf{I}_{\pi,g}$) according to the fixed order k to form a combined individual, denoted by $I_{c,g,k}$:

$$I_{c,g,k}=\{I_{\tau,g,k},I_{\pi,g,k}\} \tag{27}$$

It's worth noting that, in the study of Potter *et al.* [34], they proposed to link each individual in the current population with the best-performing individuals from the rest of the populations. However, this method is not suitable for this work as the two types of individuals are not independently optimized for separate fitness metrics but must work together to determine the common fitness metric. Therefore, we adopted the fixed linkage which makes either individual evolve in accordance with its counterpart, hence enables synergy between them.

Joint Fitness Evaluation. Each generation of population evolves by means of reproducing offspring to form the next generation. This is done by first evaluating the *fitness* of each combined individual in the population which indicates the goodness of each candidate solution in the problem domain. Fitness is determined by both bitrate and playback adaptation algorithms in this work, so it should be jointly evaluated upon a pair of the individuals, i.e., the combination in (27).

This presents a challenge as the fitness of the adaptation algorithm not only depends on the adaptation logic, but is also affected by the network conditions as well as the evaluation metric adopted. To tackle this challenge, we propose to employ trace-driven simulations to evaluate the fitness of a given combined individual according to a given fitness function (e.g., (6)). To ensure that the fitness evaluation covers a broad range of network conditions, each combined individual is evaluated over L (e.g., L=200) streaming sessions using throughput traces from a dataset $\mathbf{E}$. Now given the throughput trace data of session j in dataset $\mathbf{E}$, denoted by $S_j\in\mathbf{E}$, the combined individual $I_{c,g,k}$ can be executed (denoted by the function $F(\cdot)$) to produce a set of performance metrics (e.g., bitrate, rebuffering duration, playback latency and etc.), collectively denoted by $P_{k,j}$:

$$P_{k,j}=F(I_{c,g,k},S_j) \tag{28}$$

Finally, the fitness of $I_{c,g,k}$, denoted by $f_{c,g,k}$, is computed from the mean of all L streaming sessions:

$$f_{c,g,k}=\frac{1}{L}\sum_{j=0}^{L-1}U(P_{k,j}) \tag{29}$$

where $U(\cdot)$ is the fitness (objective) function adopted (e.g., (6)).

Selection, Crossover, and Mutation. Once the fitness values for all individuals are obtained, GP performs *selection*, *crossover* and *mutation* for the bitrate and playback adaptation population separately to reproduce offspring. Selection is to pick parent individuals with good fitness. Crossover/Mutation is to explore the combination/modification of genes in the parent individuals such that the gene diversity of the offspring can be improved to broaden the solution search space. Interested readers can refer to Potter *et al.* [34] and Koza *et al.* [30] for more details.

Termination. The reproduced offspring forms the populations in the next generation and then all the processes repeat until a predefined maximum number of generation G (e.g., G=50) is reached. As the evolutionary process goes on, GP can explore a wide spectrum of candidate solutions in the solution space to progressively evolve better-performing individuals. In the end, the combined individual with the best fitness in the final populations will be selected as the adaptation algorithm for use in actual streaming sessions.

5.3 FLAS-GP

Finally, we apply FLAS to the above GP scheme (i.e., FLAS-GP) to enable flexible playback latency control. FLAS-GP is compatible with the unified FLAS framework that operates in two phases. In *distributed offline training* phase, FLAS-GP uses state quantizer (SQ) to quantify M×N latency-QoE tradeoff states, i.e., (10). For each state, FLAS-

GP runs the GP latency-aware evolutionary process separate (c.f. Section 5.2), denoted by the function $T_{GP}(\cdot)$, to evolve a particular adaptation algorithm specifically for each state:

$$A_{p,q} = T_{GP}(U_p, C_q),\ p = 0,1,...,M-1,\ q = 0,1,...,N-1 \quad (30)$$

where U_p denotes the objective function (it is called fitness function in GP and will be maximized during the evolutionary process) with latency coefficient ω_p (defined in (6)), C_q denotes throughput trace data in throughput level ϖ_q (defined in (9)), and $A_{p,q}$ is the evolved algorithm set including a total of $M \times N$ adaptation algorithms (i.e., expression trees in GP). It is worth noting that GP is only one candidate scheme to carry out FLAS's training phase, i.e., (11), and other machine learning or heuristic paradigms can be operated by FLAS in a similar manner.

Online algorithm Selection phase of FLAS-GP is identical to that in Section 4.2. In particular, the video player does not need any GP evolutionary components or expensive computational operations online. The only modification needed for deploying FLAS-GP is to append a lightweight module into the video player to determine the runtime state of each epoch and then apply the matching algorithms to them, i.e., (12)~(20). Overall, in the two-phase design, most of the computations are completed in offline training, and a simple strategy is kept online, so that FLAS-GP can be readily implemented into real streaming platforms.

6 Performance Evaluation

In this section, we conduct a systematic and thorough evaluation for FLAS and compare it against the state-of-the-art streaming algorithms.

6.1 Evaluation Setup

We employed trace-driven simulations where the simulation settings are consistent with that in Section 3 (see Table 1) and the configurations of FLAS are summarized in Table 2. For the GP evolutionary process, we adopted a population size of 800 (i.e., 800 combined individuals), and then the population was evolved for 50 generations after which the combined individual with the best fitness was selected as the adaptation algorithm for use in actual streaming sessions.

6.2 Latency-QoE Tradeoff

Among all the streaming algorithms evaluated, three of them support the control of the target playback latency, namely FLAS-GP, FLAS-L2AC, and LAPAS. The first experiment is to evaluate how well the three algorithms track the target latency. The results in Table 4 show that the three algorithms perform similarly and the actual latency achieved is close to the corresponding target one. To further quantify the deviation of the actual latency from the target, we defined a new metric –*Mean Absolute Deviation of latency* that characterizes the average absolute difference between the actual latency and the target (henceforth called "*latency-MAD*"):

TABLE 4 Actual Mean Latency (s) vs Target Latency

	Target Latency (s)				
Algorithm	1	3	5	7	9
FLAS-GP	0.86	2.93	4.87	6.91	8.98
FLAS-L2AC	0.96	2.92	4.99	6.87	8.84
LAPAS	1.05	2.83	5.32	7.40	9.33

TABLE 5 Comparison of Latency-MAD (s)

	Target Latency (s)				
Algorithm	1	3	5	7	9
FLAS-GP	0.30	0.37	0.43	0.47	0.48
FLAS-L2AC	0.31	0.36	0.41	0.45	0.50
LAPAS	0.56	0.65	0.75	0.87	0.98

TABLE 6 QoE Performance Across Throughput Levels (Target Latency=2s)

	Throughput Level				
Algorithm	0~1	2~3	4~5	6~7	8~9
FLAS-GP	0.64	2.04	3.47	5.31	7.76
FLAS-L2AC	-1.11	1.72	3.06	4.83	6.97
LAPAS	0.21	1.79	2.79	4.07	5.20

$$\chi = \frac{1}{L}\sum_{j=0}^{L-1} |\beta_j - \lambda| \quad (31)$$

where β_j is the actual mean latency during streaming epoch j, λ is the target latency and L is the total number of streamed epochs. The results are summarized in Table 5 where lower latency-MAD indicates the target latency being better tracked. We can see that the two FLAS-optimized algorithms achieve significantly lower latency-MAD than LAPAS. This benefits from FLAS's online algorithm selection which periodically adjusts the algorithm in operation to avoid the actual latency deviating from the target during streaming a live session. In contrast, LAPAS does not have such a mechanism to make adjustments during a session. Moreover, although FLAS-L2AC and FLAS-GP perform similarly in latency-MAD, skipping video data (adopted by FLAS-L2AC) introduces playback glitches thus causing QoE degradation, while changing playback rate within 5% (adopted by FLAS-GP) is imperceptible to viewers so it is harmless to QoE. Therefore, FLAS-GP can achieve better QoE performance (see the following results).

Next, we investigate the tradeoff performance between QoE and playback latency. We observed in Fig. 10 that FLAS-GP achieves a continuous frontier of latency-QoE tradeoff where QoE is much higher than all other algorithms across the latency ranging from 1.0s to 9.0s. Table 6 summarizes the QoE achieved by FLAS-GP, FLAS-L2AC and LAPAS across throughput level 0~9. We only listed the results under 2s target latency, as similar results were obtained with other target options.

Remarkably, FLAS-GP resolves the performance flaw of FLAS-L2AC, outperforming the other two algorithms significantly at throughput level 0~1. In addition, FLAS-GP also achieves substantially higher QoE in throughput level 2~9. This clearly demonstrates that with the flexible expression tree structure, GP enables FLAS to generate more specialized algorithms to match different network conditions better.

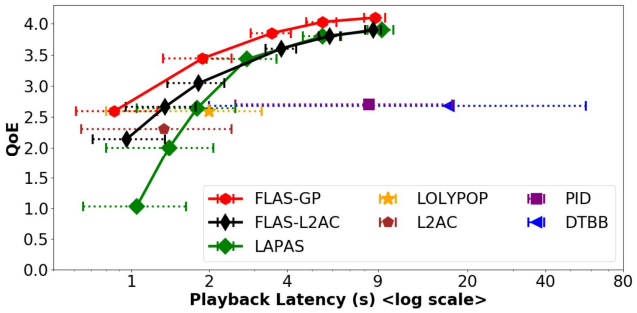

Fig. 10. Comparison of latency-QoE tradeoff under QoE function (1) (error bars span streaming sessions with top/bottom 5% latency).

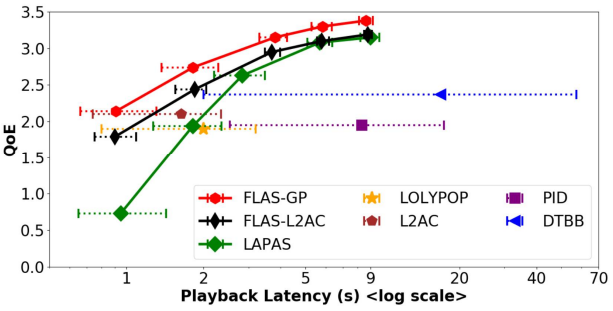

Fig. 11. Comparison of latency-QoE tradeoff under QoE function (32) (error bars span streaming sessions with top/bottom 5% latency).

The insight from the above results is that, despite the substantial performance gains in FLAS-L2AC, merely applying FLAS may not be sufficient on its own. FLAS also needs an underlying scheme that can fully exploit specializations to explore and match the wide range of network environments for optimal performance, and the GP scheme presented in Section 5 offers such an option. More work is warranted to explore other machine learning or heuristic schemes to see if one can push the envelope even further.

To see if the above observations are consistent under different QoE metrics, we repeated the experiments with another QoE function proposed by Mao *et al.* [17]:

$$Q' = \frac{1}{K}\left(\sum_{k=0}^{K-1}\vartheta_k - \sum_{k=1}^{K-1}\left|\vartheta_k - \vartheta_{k-1}\right| - 2.66\times Z - 0.2\times G\right) \quad (32)$$

where Z is the playback rebuffering duration, G is the skipped video duration, K is the total number of segments in one streaming session and

$$\vartheta_k = \log(r_k / r_{\min}) \quad (33)$$

where r_k is the bitrate selected for segment k and $r_{\min}$ is the lowest available bitrate in the profile.

Fig. 11 plots the latency-QoE tradeoff under this QoE function. We can observe very similar patterns in the results which are consistent with the observations in Fig. 10. FLAS-GP again consistently outperforms all other algorithms, more so at the lower end of the latency range.

6.3 Robustness

In this section, we investigate the robustness of FLAS. Again, we only show the performance under 2s target latency, as similar results were obtained with other options. We first consider temporal robustness – performance variation over time (i.e., days). Fig. 12 plots the daily latency-MAD (defined by (31)) over a period of 40 days. As expected, the latency-MAD of LAPAS is much higher than that of FLAS and exhibits far more fluctuations due to the changing network condition (e.g., daily throughput variations) over the 40 days. We also plotted the corresponding daily mean QoE in Fig. 12. FLAS-GP outperforms FLAS-L2AC and LAPAS substantially, achieving the highest daily QoE in 39 of the 40 days.

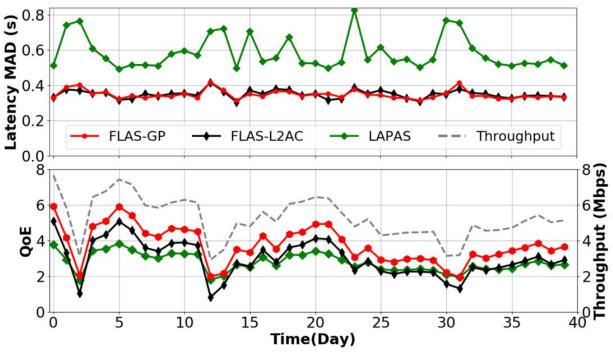

Fig. 12. Comparison of latency-MAD and QoE over a period of 40 days (target latency = 2s).

Next, we consider spatial robustness – performance variations over different network characteristics, e.g., network types, service providers and etc., which presumably exhibit different ranges of network conditions [38]. We conducted a new set of simulations using a total of seven independent throughput trace datasets, i.e., #1~#7. Table 7 summarizes the key statistics for the seven datasets where #1 to #5 were captured in 3G networks, and #6 and #7 were obtained from 4G/LTE and Wi-Fi networks respectively [36,39,40].

In order to explore the impact of trace-data usage mode in the training, we experimented with two training methods for FLAS: (a) we trained FLAS using 60 days' trace data from dataset #1 only, which is indicated by the "-D1" suffix (e.g., "FLAS-GP-D1"); and (b) we trained FLAS using 60 days' trace data consisting of the data in #1~#7, which is indicated by the "-Mix" suffix (e.g., "FLAS-GP-Mix"). In both cases, unseen trace data were used to obtain the performance results. Table 8 and Table 9 summarize the latency-MAD and QoE respectively achieved by FLAS-GP, FLAS-L2AC and LAPAS under the seven datasets.

FLAS achieves far more precise latency and better QoE performance than LAPAS. Noticeably, FLAS-GP outperforms FLAS-L2AC in QoE by 6.3%~18.1% across the seven datasets, which again exhibits the superiority of the GP scheme in live video streaming. More interestingly, FLAS trained with "-D1" and "-Mixed" perform similarly across the seven datasets despite being trained using very different trace data (e.g., the LTE network #6 has much higher mean throughput than 3G network #1). This indicates that FLAS is spatially robust.

TABLE 7 Statistics of Seven Throughput Trace Datasets

Features	Datasets						
	#1	#2	#3	#4	#5	#6	#7
Throughput (Mbps)	5.6	4.7	3.3	2.9	1.2	11.1	3.1
Variation (CoV)	0.4	0.4	0.7	0.5	0.8	0.7	0.6
Network type	3G	3G	3G	3G	3G	LTE	WiFi
Location	L1	L1	L2	L3	L4	L5	L6
ISP	S1	S2	S1	S1	S3	S2	S4

TABLE 8 Latency-MAD (s) Across Seven Datasets (Target Latency=2s)

Algorithm	Datasets						
	#1	#2	#3	#4	#5	#6	#7
FLAS-GP-D1	0.35	0.24	0.40	0.35	0.43	0.34	0.36
FLAS-GP-Mix	0.34	0.22	0.41	0.35	0.44	0.35	0.37
FLAS-L2AC-D1	0.34	0.24	0.40	0.36	0.43	0.35	0.36
FLAS-L2AC-Mix	0.35	0.23	0.39	0.34	0.44	0.33	0.33
LAPAS	0.63	0.45	0.74	0.67	0.88	0.57	0.63

Overall, the above results point to an important characteristic of FLAS – it is both temporally and spatially robust. This strongly suggests that as long as FLAS is trained with a wide spectrum of network conditions, the resultant algorithm set would be sufficiently general that could be applied to a much wider range of networks. Moreover, as the variations in network conditions have already been accounted for by the design of FLAS, it is not necessary to repeat the training process at all (unless new networks with completely different features are introduced). This can greatly save the streaming server's computing resources and simplify system deployment.

6.4 Sensitivity Analysis

In this section, we analyze the sensitivity of FLAS with respect to epoch duration and live event duration. FLAS executes algorithm selection at the beginning of each epoch in the online phase. This leads to the question of epoch duration choices. Table 10 compares the QoE of FLAS-GP across epoch durations ranging from 50s to 600s. Again only the results with 2s target latency were listed. Clearly, a longer epoch can result in better QoE performance. This is because each video epoch is regarded as a separate streaming session and longer epoch duration offers more room (i.e., time) for the adaptation algorithms to maneuver so that they do not need to be too conservative.

However, longer epoch does have a tradeoff – higher latency-MAD, as demonstrated in Table 10. We observed that the MAD increases with longer epoch durations because longer epochs reduce the execution frequency of FLAS's online algorithm selection, thereby hampering the client's responsiveness to the changes in network conditions. Meanwhile, the QoE improvement tapers off for the epoch duration longer than 300s, so 300s is adopted as the default epoch duration in this work.

Live events can have a very wide range of durations, ranging from minutes to hours. Another advantage of epoch-based FLAS is that the trained algorithm set is decoupled from the actual live event duration.

TABLE 9 QoE Performance Across Seven Datasets (Target Latency=2s)

Algorithm	Datasets						
	#1	#2	#3	#4	#5	#6	#7
FLAS-GP-D1	3.54	2.83	1.90	2.15	0.90	9.02	2.39
FLAS-GP-Mix	3.46	2.91	1.93	2.21	0.87	9.06	2.43
FLAS-L2AC-D1	3.04	2.61	1.63	1.91	0.81	8.01	2.13
FLAS-L2AC-Mix	3.01	2.66	1.71	1.93	0.75	7.87	2.09
LAPAS	2.64	1.92	1.40	1.60	0.77	6.73	1.84

TABLE 10 Impact of Epoch Duration (Target Latency = 2s).

Epoch duration (s)	50	100	300	600
QoE	3.00	3.29	3.44	3.46
Latency-MAD (s)	0.22	0.30	0.39	0.58

TABLE 11 Impact of Live Event Duration (Target Latency = 2s)

Live event duration	5 mins	10 mins	30 mins	1 hour	6 hours	24 hours
QoE	3.33	3.38	3.41	3.44	3.43	3.41
Latency-MAD (s)	0.41	0.40	0.39	0.39	0.37	0.39

For example, Table 11 shows the QoE and latency-MAD for live event durations from 5 mins all the way up to 24 hours where FLAS-GP maintains consistent QoE and low latency deviations in all cases. This strongly suggests that FLAS can be applied to a wide range of live streaming services from short-term events (e.g., live sports, live shows) to round-the-clock services (e.g., news channels and video surveillance).

7 Summaries and Future Work

The FLAS framework investigated in this paper offers a new approach to flexible latency-QoE tradeoff control for live streaming services. It not only enables precise control of playback latency all the way down to 1s, but also can achieve substantially better QoE performance than the state-of-the-art streaming algorithms. Moreover, FLAS exhibits remarkable robustness over time, mobile operators, and even network types, thereby significantly reducing the need to train streaming algorithms repeatedly. Its client-side implementation is relatively simple and does not contain any computationally intensive operations. Therefore, FLAS can be readily implemented within the current DASH/CMAF standards, offering service providers a new tool for high-performance live video streaming.

This work is only the first step in this direction. There are many opportunities for future research. For instance, as the unified FLAS framework is decoupled from the underlying streaming algorithms, it means that one can replace the later to explore the use of other machine learning or heuristic paradigms to further improve the performance. On the other hand, exploring FLAS's robustness in a wider range of network types (e.g., the emerging 5G or datacenter networks) is also a fruitful direction for future work.

REFERENCES

[1] *Youtube Live* [Online] https://www.youtube.com/channel/UC4R8DWoMoI7CAwX8_LjQHig

[2] *Facebook Live* [Online] https://live.fb.com/

[3] *Latency Options of YouTube Live.* [Online] https://support.google.com/youtube/answer/7444635?hl=en

[4] *Latency Options of Twitch.* [Online] https://help.twitch.tv/s/article/low-latency-video?language=en_US

[5] *Latency Options of Amazon Web Services for Live Streaming.* [Online] https://aws.amazon.com/media/tech/video-latency-in-live-streaming/?nc1=h_ls

[6] G. Zhang and J.Y.B. Lee, "Ensemble Adaptive Streaming-A New Paradigm to Generate Streaming Algorithms via Specializations," *IEEE Transactions on Mobile Computing*, Apr 2019.

[7] C. Liu, I. Bouazizi and M. Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," *Proc. ACM Conf. Multimedia Syst. (MMSys'11)*, San Jose, USA, Feb. 2011, pp.169-174.

[8] J. Jiang, V. Sekar and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE," *Proc. Conf. Emerging Networking Experiments and Technologies (CoNEXT'12)*, Nice, France, Dec. 2012, pp.97-108.

[9] T.Y. Huang, R. Johari, N. McKeown, M. Trunnell and M. Watson, "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," *Proc. ACM SIGCOMM*, Chicago, Illinois, USA, Aug. 2014, pp.187-198.

[10] K. Spiteri, R. Urgaonkar and R.K. Sitaraman, "BOLA: Near-Optimal Bitrate Adaptation for Online Videos," *Proc. IEEE INFOCOM*, San Francisco, CA, USA, Apr 2016, pp.1-9.

[11] X. Yin, A. Jindal, V. Sekar and B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," *Proc. ACM SIGCOMM*, London, United Kingdom, Aug 2015, pp.325-338.

[12] A.H. Zahran, D. Raca, C. Sreenan. "ARBITER+: Adaptive Rate-Based InTElligent HTTP StReaming Algorithm for Mobile Networks". *IEEE Transactions on Mobile Computing*, vol.17(12), Apr 2018, pp.2716-2728.

[13] Y. Qin, R. Jin, S. Hao, K.R. Pattipati, F. Qian, S. Sen, B. Wang and C. Yue, "A Control Theoretic Approach to ABR Video Streaming: A Fresh Look at PID-based Rate Adaptation," *Proc. IEEE INFOCOM*, Atlanta, GA, USA, May 2017, pp.1-9.

[14] Y. Liu and J.Y.B. Lee. "A Unified Framework for Automatic Quality-of-Experience Optimization in Mobile Video Streaming," *Proc. IEEE INFOCOM*, San Francisco, CA, USA., Apr 2016, pp.1-9.

[15] Y. Liu and J.Y.B. Lee, "Post-Streaming Rate Analysis - A New Approach to Mobile Video Streaming with Predictable Performance," *IEEE Transactions on Mobile Computing*, vol.16(12), Dec 2017, pp.3488-3501.

[16] Z. Akhtar, Y.S. Nam, R. Govindan, "Oboe: Auto-tuning Video ABR Algorithms to Network Conditions" *Proc. ACM SIGCOMM*, Budapest, Hungary, Aug 2018, pp.44-58.

[17] H. Mao, R. Netravali, M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, Aug 2017, pp.197-210.

[18] F. Chiariotti, S. D'Aronco, L. Toni, "Online Learning Adaptation Strategy for DASH Clients," *Proc. ACM Multimedia Syst.*, Klagenfurt, Austria, May 2016, pp.8:1-8:12.

[19] V. Martín, J. Cabrera, N. García, "Design, Optimization and Evaluation of a Q-Learning HTTP Adaptive Streaming Client," *IEEE Transactions on Consumer Electronics*, vol. 62(4), Nov 2016, pp.380-388.

[20] M. Gadaleta, F. Chiariotti, M. Rossi, A. Zanella, "D-DASH: A Deep Q-learning Framework for DASH Video Streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3(4), Dec 2017, pp.703-718.

[21] N. Chen, S. Quan, S. Zhang et. al, "Cuttlefish: Neural Configuration Adaptation for Video Analysis in Live Augmented Reality," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32(4), April 2021, pp. 830-841.

[22] L. De Cicco, S. Mascolo, V. Palmisano, "Feedback Control for Adaptive Live Video Streaming," *Proc. ACM Multimedia Syst.*, San Jose, USA, Feb 2011, pp.145-156.

[23] J. Wang, S. Meng, J. Sun, Z. Quo, "A General PID-based Rate Adaptation Approach for TCP-based Live Streaming over Mobile Networks," *Proc. IEEE ICME*, Seattle, USA, Jul 2016, pp.1-6.

[24] L. Xie, C. Zhou, X. Zhang, "Dynamic Threshold Based Rate Adaptation for HTTP Live Streaming," *Proc. IEEE ISCAS*, Baltimore, MD, USA, May 2017, pp.1-4.

[25] K. Miller, A.K. Al-Tamimi, A. Wolisz, "QoE-Based Low-delay Live Streaming Using Throughput Predictions," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 13(1), Jan 2017, pp.1-24.

[26] M. Lim, M.N. Akcay, A. Bentaleb, "When They Go High, We Go Low: Low-latency Live Streaming in Dash.js with LoL," *Proc. ACM Multimedia Systems Conference*, Istanbul, Turkey, May 2020, pp. 321-326.

[27] Y. Zhao, Q.W. Shen, W. Li, "Latency Aware Adaptive Video Streaming using Ensemble Deep Reinforcement Learning," *Proc. ACM International Conference on Multimedia*, Nice, France, Oct 2019, pp.2647-2651.

[28] G. Zhang and J.Y.B. Lee, "LAPAS: Latency-Aware Playback-Adaptive Streaming," *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakech, Morocco, April 2019, pp.1-6.

[29] V. Mnih, A.P. Badia, M. Mirza, A. Graves, "Asynchronous Methods for Deep Reinforcement Learning," *Proc. International Conference on Machine Learning*, New York City, NY, USA, Jun 2016, pp. 1928-1937.

[30] J.R. Koza, "Genetic Programming as A Means for Programming Computers by Natural Selection," *Springer Statistics and computing*, vol. 4(2), Jun 1994, pp.87-112.

[31] L. Golubchik, John C.S. Lui and R.R. Muntz, "Reducing I/O Demands in Video-on-Demand Storage Servers," *Proc. ACM SIGMETRICS*, Ottawa, Canada, May 1995, pp.25-36.

[32] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," *Proc. ACM Multimedia Syst.*, San Jose, USA, Feb 2011, pp.133-144.

[33] *Common Media Application Format (CMAF)* [Online] https://mpeg.chiariglione.org/standards/mpeg-a/common-media-application-format

[34] M.A. Potter, K.A.D. Jong, "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents," *Evolutionary Computation*, vol. 8(1), Spring 2000, pp.1-29.

[35] K.H. Ang, G. Chong and Y. Li, "PID Control System Analysis, Design, and Technology," *IEEE Transactions on Control Systems Technology*, vol.13(4), Jul. 2005, pp.559-576.

[36] *Mobile Throughput Trace Data.* [Online] http://sonar.mclab.info/tracedata/TCP/

[37] *Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad*, Apple Inc, retrieved on Aug 2016. [Online] https://developer.apple.com/library/ios/technotes/tn2224/_index.html

[38] G. Zhang, R.K.H. Ngan, J.Y.B. Lee, "EmuStream – An End-to-End Platform for Streaming Video Performance Measurement," *IEEE Access*, vol. 8, Dec 2019, pp.669-680.

[39] H. Riiser, P. Vigmostad, C. Griwodz, P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," *Proc. ACM Multimedia Syst.*, Oslo, Norway, Feb 2013, pp.114-118.

[40] *L2AC Source Code* [Online] https://github.com/anlanzy/L2AC

[41] *Dash.js Reference Player* [Online] https://reference.dashif.org/dash.js/latest/samples/low-latency/index.html