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 has seen tremendous growth in the past decade. 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 in changing network conditions, let alone achieve the service-desired latency-QoE tradeoff. To tackle the challenge, this article develops a novel framework called Flexible Latency Aware Streaming (FLAS) that not only can achieve consistent low latency, but also control the latency-QoE tradeoff flexibly. Specifically, FLAS generates a set of adaptation 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 optimizing the existing algorithms, then developed a novel Genetic Programming approach to fully exploit 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 and reliable solution for live-streaming services.

Index Terms—Video streaming, genetic programming, quality-of-experience, video reliability

1 Introduction

Video streaming has become a mainstream application in the 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). 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, a unique and important performance metric in live streaming is *playback latency*, which is defined as the time difference between video rendering and actual capturing. In this paper, latency is not included in the calculation of QoE in order to differentiate it from the usual QoE metrics.

In general, live streaming requires low playback latency (a few seconds at most), otherwise the service quality would degrade significantly. In fact, however, latency and QoE (e.g., video quality) are inherently conflicting objectives. For instance, viewers generally prefer to stream high-quality videos which would inevitably incur longer transmission delay at the mobile radio link. As the transmission delay translates directly into the playback latency, the need for high-quality videos inherently conflicts with the low latency demand. Therefore, this requires live streaming services to seek performance tradeoffs between the latency and QoE.

In practice, different types of live streaming services can have very different latency-QoE requirements [3], [4], [5]. For example, highly interactive live streams (e.g., live sales, interactive live shows) demand much shorter latency than one-way broadcasts (e.g., news, concerts), but the interactive ones can tolerate lower video quality. Therefore, how to achieve *desired* and *optimal* latency-QoE tradeoffs for different live services is a significant challenge for designing streaming algorithms.

While many sophisticated live streaming algorithms (e.g., [22], [23], [24], [25], [26], [27], [28]) were proposed in recent years, none of them have addressed the above challenge. Worst still, 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 to keep

- *Guanghui Zhang is with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong, and also with the Centre for Advances in Reliability and Safety (CAiRS), Pak Shek Kok, NT, Hong Kong. E-mail: ghzhang@link.cuhk.edu.hk.*
- *Jack Y. B. Lee is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong. E-mail: yblee@ie.cuhk.edu.hk.*
- *Ke Liu is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100732, China, and also with the University of Chinese Academy of Sciences (UCAS), Beijing 100049, China. E-mail: liuke@ict.ac.cn.*
- *Haibo Hu is with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong, and also with the PolyU Shenzhen Research Institute, Kowloon, Hong Kong. E-mail: haibo.hu@polyu.edu.hk.*
- *Vaneet Aggarwal is with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA, and also with the Computer Engineering, Purdue University, West Lafayette, IN 47907 USA. E-mail: vaneet@purdue.edu.*

Manuscript received 17 Nov. 2020; revised 8 Apr. 2021; accepted 16 May 2021. Date of publication 24 May 2021; date of current version 11 June 2021.
(Corresponding author: Ke Liu.)
Recommended for acceptance by T. Kosar.
Digital Object Identifier no. 10.1109/TPDS.2021.3083202

consistent latency, let alone achieve the desired latency-QoE performance.

To tackle the challenge, we propose a novel framework called Flexible Latency Aware Streaming (FLAS) that not only can achieve consistent low latency, but also control the latency-QoE tradeoff flexibly. Specifically, FLAS introduces the notion of *state quantizer* (SQ) to quantify the latency-QoE tradeoff, and then generates a set of adaptation logics offline where each one is optimized for a candidate tradeoff point. At runtime, service providers (or viewers) are allowed to specify target playback latency (e.g., 2s) according to the service requirement. With the target latency prescribed, FLAS then periodically selects/adjusts the operational adaptation logic based on the network condition, to ensure the actual latency not deviate from the target while QoE can be maximized.

This work makes three key *contributions*. First, we conducted extensive evaluations to reveal the problems of the existing live streaming algorithms. Second, we proposed FLAS, which is a unified framework that can optimize the existing learning-based algorithms. To demonstrate this applicability, we applied FLAS to L2AC [27], which is a state-of-the-art algorithm developed upon A3C [29]. However, we discovered that although FLAS-L2AC can achieve substantially better performance, its efficacy is restricted under challenging network conditions, presumably due to the inherent neural network structure or the training parameters. Third, to get rid of the limitation, we turned to using a radically different machine-learning approach – Genetic Programming (GP) [30]. Different from deep reinforcement learning, GP represents candidate solutions in the form of expression trees and does not impose any rigid structures on the tree, so the solution space can be explored freely. Based on this property, we improved GP evolutionary process to make it compatible with FLAS.

Extensive evaluations show that FLAS-GP can fully exploit FLAS's potential. Specifically, 1) FLAS-GP can achieve substantially higher QoE with the same or lower playback latency than the state-of-the-arts; 2) it can precisely control the latency all the way down to 1s; 3) it exhibits remarkable spatial and temporal robustness; and 4) it 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 evaluates the existing live streaming algorithms; Section 4 proposes FLAS and evaluates its efficacy by FLAS-L2AC; Section 5 presents a novel Genetic Programming approach; Section 6 compares FLAS with the state-of-the-arts, and Section 7 summarizes the study and outlines future work.

2 Background and Related Work

Dynamic adaptive video streaming (DASH [32]) is a primary tool service providers use to compensate for the inevitable bandwidth fluctuation in the network. In recent years, researchers have developed many novel adaptive streaming algorithms for on-demand streaming. The basic principle is to dynamically select the future video bitrate in the light of past measurements such as throughput and buffer occupancy. Existing adaptive 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], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21] approaches, or classified according to the technique used to optimize the adaptation logic, e.g., heuristics [7], [8], [9], [10], [11], [12], feedback control [13], data-analytic [14], [15], [16], and machine learning [17], [18], [19], [20], [21].

While DASH works well for on-demand streaming, due to its segment-based video transmission, it exhibits latency too long to be suited for live streaming. To tackle this issue, the recently proposed CMAF [40], [43] supports to divide the DASH segments into independently transferable chunks, which fuels the researchers to develop new adaptation algorithms for live streaming.

First, since data buffering can directly increase the 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 at the streaming server side. Xie *et al.* [24] proposed DTBB to select video bitrate depending on a buffer threshold that can be dynamically tuned based on the measured throughput.

The second problem in live streaming is called latency *accumulation*. Specifically, during rebuffering where the player runs out of the 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 event. Given that video data is played sequentially, the latency cannot be reduced once introduced, resulting in incremental latency whenever a rebuffering event occurs.

Two common solutions to this problem are video 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 playback. Both of them can make the player catch up with the live event. For instance, Miller *et al.* [25] proposed LOLYPOP that executes video skipping once the playback latency is larger than a pre-defined skipping threshold. Lim *et al.* [26] proposed LoL that turns up the playback rate to lower latency. Zhao *et al.* [27] developed L2AC that incorporates both of the methods by training neural networks. Although the above algorithms were designed with reducing latency in mind, it's still far from enough in practice, as different live services can have very different latency requirements. Recently, Zhang *et al.* [28] proposed LAPAS that can achieve tunable latency control through analyzing past network trace data. However, LAPAS suffers from several fundamental limitations, which will be discussed in Section 3.

By contrast, the FLAS framework developed in this study offers three superiorities: 1) In addition to achieving consistent low latency over different network conditions, FLAS can enable flexible latency control; 2) FLAS is a general framework, which not only can be applied to optimizing the existing algorithms, but also guide the design of new latency-aware approaches. For instance, we applied FLAS

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]
Live event duration	3600s
Initial video bitrate	0.2 Mbps
Prefetching duration	0.04s, i.e., 1 frame

TABLE 2
Statistics of Seven Throughput Trace Datasets

	Datasets						
Features	#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
Mobile operator	S1	S2	S1	S1	S3	S2	S4
Collection location	L1	L1	L2	L3	L4	L5	L6

to L2AC [27] in Section 4, and explored a novel Genetic Programming approach in Section 5. This feature enables FLAS to incorporate any advanced techniques to fully liberate its performance potential; 3) The two-phase design of FLAS avoids the difficulties of real deployment, bringing a completely practical solution to live streaming.

3 Evaluation of Existing Algorithms

In this section, we evaluate the performance of five leading live streaming algorithms and then discuss their limitations.

3.1 Experiment Setup

To evaluate the streaming algorithms in realistic network settings, we employed an open-source trace-driven simulator developed by Yi *et al.* [41], which supports frame-basis video transmission [40], [43]. We refer readers to the paper [41] and the source code [33] for more details. In our evaluation, we changed the value of a few parameters in the simulator, which were listed in Table 1. Five state-of-the-art live streaming algorithms were evaluated: PID [23], DTBB [24], LOLYPOP [25], L2AC [27], LAPAS [28].

The network conditions were emulated by replaying TCP throughput trace data captured from real mobile networks. The statistics of the trace data used in the evaluation were summarized in Table 2, where #5 was captured by Riiser *et al.* [38], #7 was provided by the simulator [33] and the rest (#1∼#4, and #6) were captured by us [36] (see Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2021.3083202, for the throughput distribution). In the rest of the paper, unless stated otherwise, the trace data used for evaluations are composed of the dataset #1 to #7.

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 [41]:

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

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, and the weights of these components (e.g., 3.0, 0.2) follow [11] and [41]. Moreover, we will further consider other QoE metrics in Section 6.1.

3.2 Results and Discussions

Observation 1. Fig. 1 compares the mean value of QoE and playback latency of the five streaming algorithms. We can see that most algorithms are *not* latency tunable, so each achieves only one specific tradeoff point between QoE and latency. This is a significant limitation in practice as different types of live streaming services can have very different latency/QoE requirements. By comparison, LAPAS obtains a continuous tradeoff trajectory, as LAPAS supports the configuration of target latency and adopts a streaming-parameter-optimization to track the target. Interested readers can refer to [28] for more details.

However, under the same latency, the QoE performance of LAPAS is much worse than L2AC (a more comprehensive QoE comparison is in Appendix A.2, available in the online supplemental material). We argue that this is because LAPAS's adaptation logic is a pre-programmed fixed heuristic, which is inevitably restricted by the human intuitions. In comparison, L2AC employs deep reinforcement learning (i.e., A3C [29]) that learns a better adaptation logic based on its past experience. Nonetheless, as we mentioned, L2AC is not a latency-tunable algorithm, so it is incapable to achieve desired latency-QoE tradeoffs based on the user's requirements.

Observation 2. In Fig. 1, the error bars indicate the latency range achieved in different streaming sessions by each algorithm. We found that the latency across different sessions is far from consistent, but varies substantially. To further investigate this issue, we plotted Fig. 2 to show latency variations over a period of 40 days by using a 40-day TCP throughput trace from dataset #1. To appreciate the variations in network conditions, we also plotted the daily mean TCP throughput in Fig. 2, which fluctuates significantly from a low of 3.3 Mbps to a high of 7.8 Mbps. As expected, the daily latency of most algorithms (except LAPAS) fluctuate substantially under the changing network conditions, e.g., the latency of DTBB ranges from 2s to 39s, which is clearly undesirable in practice.

By contrast, LAPAS exhibits more consistent latency over the 40 days (we set target latency to 1s). This is because LAPAS optimizes/tunes the streaming parameter periodically through analyzing past throughput trace data to adapt to the changing network conditions. However, this is at the expense of extremely high computational complexity. Specifically, the video clients of LAPAS need to keep detecting the network condition, collecting throughput trace data, and then feedback to the streaming server. Based on the trace data, the server periodically (daily) updates/optimizes the parameters of the adaptation algorithm to adapt to the changing network condition. As the network condition may differ across different clients, the parameter optimization is

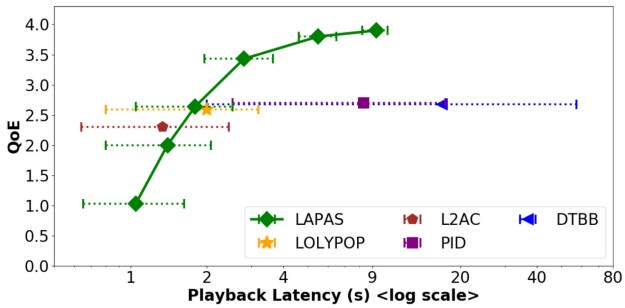


Fig. 1. Comparison of QoE and playback latency (error bars span the streaming session with top/bottom 10 percent latency).

conducted separately for each client. Therefore, as the client number increases, the computational overhead increases dramatically. Based on our measurement, one-day optimization for 100 clients even costs ~1320 CPU hours, which undoubtedly occupies a large amount of computational resource when the client scales, and hinders the large-scale deployment on the real streaming platforms.

4 Flexible Latency Aware Streaming

To tackle the limitations of the existing algorithms, we develop a unified framework called Flexible Latency Aware Streaming (FLAS). FLAS is built upon two phases, namely *distributed offline training* and *online adaptation logic selection*, as depicted in Fig. 3. In this section, we first introduce the design of FLAS and then evaluate its efficacy by applying it to optimizing an existing algorithm L2AC [27].

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

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

where the first dimension $\boldsymbol{\omega}$ is called *latency coefficient* and the second dimension $\boldsymbol{\varpi}$ is named as *throughput level*.

Latency Coefficient $\boldsymbol{\omega}$. The function is to extract the relationship between playback latency and QoE. Specifically, let $\boldsymbol{\mu}_k$ 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 $\boldsymbol{\alpha}_k$ can be computed from

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

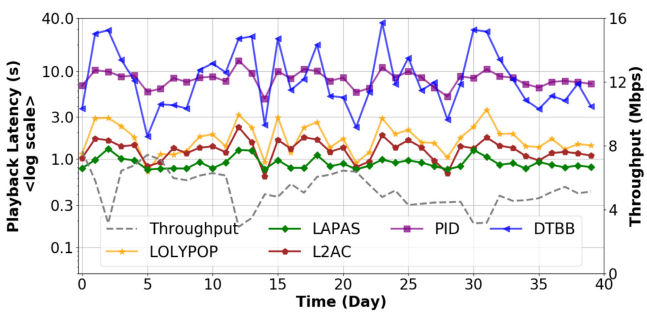


Fig. 2. Comparison of daily latency over a period of 40 days.

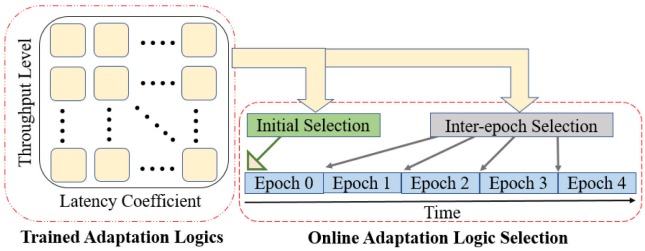


Fig. 3. The system architecture of FLAS.

The mean playback latency over different video segments is given by

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

where K is the total number of segments in an epoch[1]. The mean latency $\boldsymbol{\beta}$ is then combined with the QoE function, denoted by Q (e.g., (1)), to form the objective function for the training:

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

where $\boldsymbol{\omega}$ is the *latency coefficient*.

The objective function U will then be maximized during the training. As the playback latency $\boldsymbol{\beta}$ and QoE Q are conflicting metrics, the latency coefficient $\boldsymbol{\omega}$ can be tuned to control the latency-QoE tradeoff. For instance, a larger value of $\boldsymbol{\omega}$ will train an adaptation logic with lower latency and worse QoE. Based on this principle, we define M values of $\boldsymbol{\omega}$, i.e., $\{\boldsymbol{\omega}_p \,|\, p = 01,\ldots,M\text{-}1\}$, to generate M different objective functions:

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

so that we can quantify M candidate latency-QoE tradeoff points.

Throughput Level $\boldsymbol{\varpi}$. To tackle the latency variations across different network conditions, we differentiate the network conditions through throughput level, which is defined as the mean throughput of each epoch (network type is also a potential differentiator, but it's too coarse-grained to perform well, c.f. Appendix A.1, available in the online supplemental material). The challenge it that although throughput level can be calculated directly offline as the trace data is given, it cannot be known before streaming the actual video online. Therefore, to keep the calculation method consistent, we propose to *estimate* the throughput level.

Specifically, the throughput of new epoch j can be estimated by the mean throughput of downloading epoch j-1:

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

1. In the process of live streaming, the network condition may change significantly, so the initial-selected adaptation logic may no longer be optimal later on. We thus propose to divide one streaming session into multiple sub-sessions, called "epoch", where each has a fixed video duration (e.g., 300s), so the system can execute adaptation logic re-selection in the interval of each epoch to adapt to the changing network. This is illustrated in Fig. 3 and we will introduce the details in Section 4.2.

where $s_{j\text{-}1,k}$, and $d_{j\text{-}1,k}$ are the size and download time of segment k, and m is the segment number. We then apply a linear quantization policy to map the measured throughput V_j to the discrete *throughput level* $\boldsymbol{\varpi}_j$:

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

where Δ is the quantization step size and N is the total step number. The next step is to segregate the throughput trace data of all epochs, S_j, $j = 01,\ldots,J$, into N network classes:

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

where each class will emulate a particular network condition for the training.

Training. With the state quantizer (SQ), FLAS obtains a total of $M \times N$ states:

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

where U_p is the objective functions (defined (6)) and C_p is the network class (defined in (9)). For each state, FLAS runs a separate training process, denoted by function $T_x(\cdot)$, to train a specialized adaptation logic:

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

where $A_{p,q}$ is the trained logic set (illustrated in Fig. 3 where each square represents an adaptation logic). During the training, FLAS also records the mean QoE, denoted by $Q_{p,q}$, and mean latency, denoted by $\boldsymbol{\beta}_{p,q}$, of each state, which will be utilized in the online phase.

4.2 Online Adaptation Logic Selection

The adaptation logics trained offline will be downloaded to the video player (through DASH metadata [32]) for online streaming. To cater to different latency requirements, FLAS supports runtime configuration of target latency (e.g., 2s), denoted by $\boldsymbol{\lambda}$, which is input as a player option. With $\boldsymbol{\lambda}$ prescribed, the system's goal is to prevent the actual latency from deviating from the target while maximizing QoE.

Initial Selection. FLAS first selects an adaptation logic to run at *epoch 0* through state quantizer (SQ). Specifically, the video client begins a live streaming session with prefetching m video segments where the total prefetching video duration equals to the target latency $\boldsymbol{\lambda}$. FLAS then measures the mean throughput in downloading these m 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. The average throughput V_0 will then be mapped to the 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 the total step number.

Then the next step is to determine latency coefficient, and the criteria is to find which logic can achieve the highest QoE while its playback latency does not exceed the target $\boldsymbol{\lambda}$:

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

where Q_{p,q^*} ($\boldsymbol{\beta}_{p,q^*}$) are the recorded QoE (latency) with latency coefficient $\boldsymbol{\omega}_p$ and throughput level $\boldsymbol{\varpi}_{q^*}$. Finally, the video player will apply the matching adaptation logic to epoch 0.

Inter-Epoch Selection. For streaming the subsequent epochs (e.g., epoch 1∼4 in Fig. 3), the network condition may change significantly, so the previously selected adaptation logic may no longer be optimal. Therefore, FLAS will adjust the operational logic at the start of each epoch to adapt to the changing network.

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

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

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 between the actual latency $\boldsymbol{\xi}_{j-1}$ and the target $\boldsymbol{\lambda}$ in epoch $j-1$.

To compensate for the latency deviation, FLAS will adjust the target latency for epoch j based on the output of the PI controller:

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

where $\boldsymbol{\lambda}_{j-1}$ is the target latency adopted in epoch $j-1$. An intuitive example to illustrate this feedback principle is that, if the actual latency is higher than the target 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 $\boldsymbol{\lambda}_j$, the next step is to select the adaptation logic using 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 $s_{j\text{-}1,k}$, and $d_{j\text{-}1,k}$ are the size and download time of segment k. The mean throughput V_j is then mapped to the throughput level by

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

Latency coefficient is determined by

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

TABLE 3
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]

which is similar to (14) except $\boldsymbol{\lambda}_j$ replacing $\boldsymbol{\lambda}$. Finally, the video player will apply the matching adaptation logic to epoch j.

4.3 Performance Evaluation

FLAS is a *general* framework that is able to optimize the existing algorithms. To evaluate this applicability, in this section, we apply FLAS to L2AC [27], where the training function $T_x(.)$ (see (11)) is deep reinforcement learning A3C [29], and the output $A_{p,q}$ are $M \times N$ neural networks. More details are in Appendix A.3, available in the online supplemental material.

We conducted the trace-driven simulation as described in Section 3. 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. Unless stated otherwise, the epoch duration is set to 300s. The rest of the parameters are summarized in Table 3.

Fig. 4 compares the latency-QoE performance of FLAS-L2AC to the existing algorithms. We observed that FLAS not only configures L2AC to offer a continuous tradeoff trajectory, but also improves the QoE significantly (a more comprehensive QoE comparison is in Appendix A.2, available in the online supplemental material). This strongly suggests that FLAS and L2AC can cooperate effectually to train more specialized neural networks to match the operating environments better. Compared to LAPAS, FLAS-L2AC achieves much higher QoE in the lower latency domain ($\leq$3s), and performs similarly in the higher latency domain (>3s).

Fig. 5 plots the daily mean latency over a period of 40 days (the trace data is from dataset #1, c.f. Table 2). We set the target latency to 1s for FLAS-L2AC and LAPAS. 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 and per-client basis optimization (discussed in Section 3), FLAS-L2AC does not need to repeat the training at all, as the latency variations largely due to network condition changes have already been addressed. Moreover, FLAS's offline training carries most of the system complexity (requires about 3718 CPU hours and can be executed in full parallel) and once it is completed no need to be re-executed, so that the deployment for FLAS is much simpler.

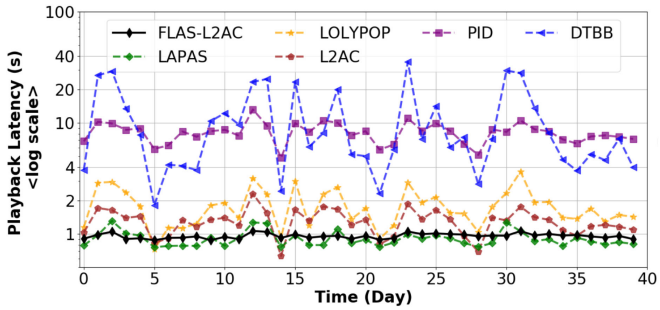


Fig. 5. Comparison of daily playback latency over 40 days.

Limitations of FLAS-L2AC. Fig. 6 compares the daily mean QoE of FLAS-L2AC and LAPAS under 1s target latency over the 40 days. We also plotted the daily mean TCP throughput to appreciate the variation of network conditions. We found that the QoE tracks the throughput closely, as the latter directly impacts the delivered video bitrate, 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 30 of the 40 days and performs worse in the rest 10 days. Referring to the daily throughput, it appears that FLAS-L2AC becomes *less effective* in networks with poor network conditions.

To this end, we further studied the QoE performance across different throughput levels. We divided all streaming sessions into 10 throughput levels, with level $l = 01,\ldots,8$ collecting sessions with average throughput within (l, l+1] Mbps, plus level 9 with average throughput $\geq$9Mbps, and then summarized the respective QoE of FLAS-L2AC and LAPAS in Table 4. The results verify our conjecture that FLAS-L2AC performs much worse in lower throughput levels, especially at the lowest two (i.e., 0$\sim$1). Similar results are 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

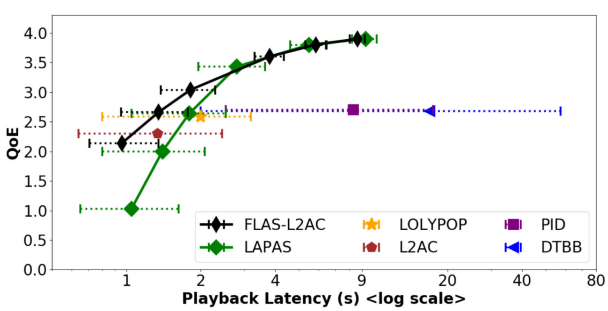


Fig. 4. Comparison of QoE and playback latency (error bars span the streaming session with top/bottom 10 percent latency).

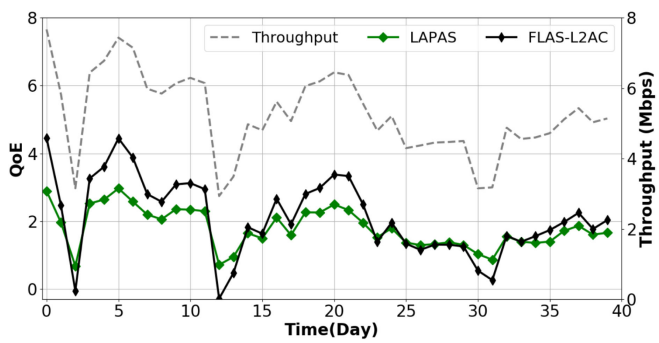


Fig. 6. Comparison of daily QoE over a period of 40 days.

TABLE 4
QoE Performance Across Throughput Levels
(Target Latency = 1s)

Algorithm	Throughput Level				
	0∼1	2∼3	4∼5	6∼7	8∼9
FLAS-L2AC	-2.02	0.82	2.02	3.92	5.21
LAPAS	-0.53	0.87	1.59	2.81	3.77

obtained. To shed light on the results in Fig. 6 and Table 4, we attempted to tackle the challenge by analyzing the adaptation logic's bitrate selection 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 the measured throughput (x-axis) for the adaptation logics.

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 that its bitrate adaptation is aggressive. While this may work well in high throughput levels with stable network conditions, it would cause disastrous consequences in low throughput levels which typically have substantial fluctuations. 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 adopted neural network structure may still not be sufficiently flexible to explore the complete solution space for the bitrate adaptation. In comparison, LAPAS exhibits reasonable conservatism at throughput level 0. However, the conservatism cannot be properly changed at higher throughput levels due to the limitations of the fixed heuristic logic, so that inevitably leads to 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) were plotted in Fig. 7 which presents a far more reasonable behavior. We observed that, as the measured throughput increases, 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

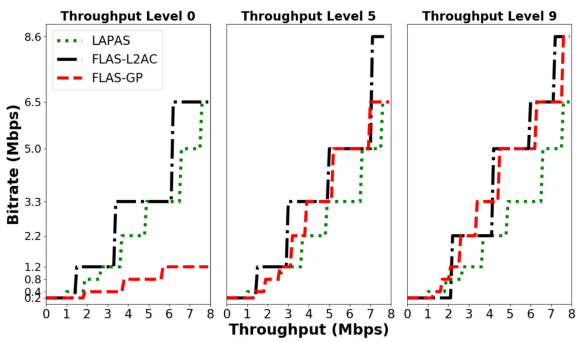


Fig. 7. Comparison of bitrate adaptation behavior in low (0), medium (5), and high (9) throughput levels (target latency = 2s).

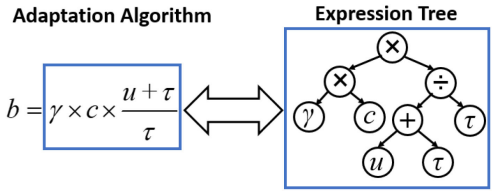


Fig. 8. Illustration of the transform between adaptation algorithm and GP expression tree (c, u are estimated throughput and buffer occupancy respectively; γ and τ are numeric constants; b is video bitrate).

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. At level 5, FLAS-GP becomes more moderate and balanced. At level 9, FLAS-GP becomes more aggressive, even occasionally selecting bitrates higher than the measured throughput. The intuition is that low measured throughput at level 9 is likely short-term so maintaining high bitrates can prevent unnecessary QoE degradations. We will introduce the design of FLAS-GP in next section.

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* [30] 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 problems of FLAS-L2AC.

In this section, we investigate the GP approach for FLAS where adaptation logics are encoded with expression trees. In addition, we improved the GP evolutionary process to make it more suitable for evolving adaptation logic in the scenario of live video streaming. It is worth noting that other machine learning, heuristic techniques, and streaming protocol can be operated by FLAS in a similar manner.

5.1 Adaptation Logic and Expression Tree

In fact, expression tree is particularly suitable for representing adaptive streaming algorithms (logics). Fig. 8 shows an example: the right-side expression tree is the equivalent of the left-side expression, which is a hybrid-throughput-buffer-based on-demand adaptation algorithm proposed by Liu *et al.* [14]. This algorithm determines 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 logics* for live streaming.

Playback Adaptation. A problem in live streaming is called *latency accumulation* and the source is playback rebuffering. Specifically, a rebuffering event occurs when the video player runs out of video data and thus has to suspend the video playback until more 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. Worst still, as the subsequent

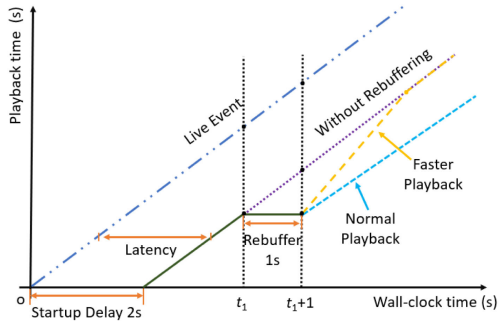


Fig. 9. Illustration of using adaptive playback to control playback latency.

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. Therefore, whenever a rebuffering event occurs, the playback latency will be increased by the rebuffering, which is clearly undesirable in live streaming.

To the best of our knowledge, there are two effective methods to address this problem. The first one is video data skipping [25]. Through 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 turn up the video playback framerate slightly (e.g., within 5 percent) 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, in this work, we employ playback rate regulation to control the latency. Fig. 9 illustrates the relationship between playback latency, rebuffering, and playback rate. 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 the live event 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 GP to evolve playback adaptation logics. Specifically, the input of an expression tree is called *variable operand*[2], which captures the network and streaming states. We define variable set $\Re$, which includes four variable operands:

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

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 streaming while the last one is specific to live video streaming.

In addition to input variables, we also need numeric constants to construct adaptation logics, which can be introduced into the expression tree via *constant operand*. We define constant set $\Im$, where the constant operands are randomly generated over a given range D:

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

As opposed to operand, the non-leaf node of the expression tree is called *operator*, which performs a specific operation on its child nodes to produce a result to serve its parent node. We define set $\aleph$ which contains four arithmetic operators:

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

The above operands and operators are specially chosen such that the resultant expression tree can be presented in the form of a mathematical equation which can be simply implemented into video players. The output of the expression tree is playback rate multiplier, denoted by θ. Specifically, $\theta = 1$ means normal playback rate and $\theta > 1$ ($\theta < 1$) speeds up (slows down) the 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], [45] found that, for both video and audio, playback rate changes within ± 5 percent are imperceivable to most viewers, so we applied a maximum change limit $\kappa_{\max}$ to the playback rate multiplier:

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

where $\kappa_{\max} = 5\%$.

Bitrate Adaptation. For evolving bitrate adaptation logics, we define variable set Φ that includes five variable operands:

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

Comparing (25) to (21), an additional operand ρ – the playback rate calculated from (24) is added. This means that the bitrate adaptation logic has knowledge of the playback rate decision to enable the two kinds of adaptation logics to evolve jointly (c.f. Section 5.2). The definitions of the constant operands and operators in bitrate adaptation are identical to (22) and (23) respectively. The output of the expression tree is video bitrate, denoted by r, but as r is a real number while bitrate choices are discrete, it needs to be mapped to the closest available bitrate version by

$$h = \arg\min_h |r_h - r|, \tag{26}$$

where r_h, $h = 01,\ldots,H$-1, are the available bitrate versions.

2. Expression trees comprise two types of components: *operands* – leaf nodes, and *operators* – non-leaf nodes. The choices of operands and operators determine GP's search space.

5.2 Latency-Aware GP Evolutionary Process

The two types of adaptation logics (for playback rate and bitrate) 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 logics, 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 logics.

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

Let $\mathbf{I}_{\boldsymbol{\tau},g}$ ($\mathbf{I}_{\pi,g}$) and $I_{\boldsymbol{\tau},g,k} \in \mathbf{I}_{\boldsymbol{\tau},g}$ ($I_{\pi,g,k} \in \mathbf{I}_{\pi,g}$) be the population set and individual k, $k = 01,\ldots,K$–1, in the population in generation g, $g = 01,\ldots,G$-1, for playback (bitrate) adaptation logics respectively. We link each pair of individuals ($I_{\boldsymbol{\tau},g,k}$ and $I_{\pi,g,k}$) from the two populations ($\mathbf{I}_{\boldsymbol{\tau},g}$ 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 individual in the other population. However, this method is not suitable for this work as the two types of individuals are not independently optimized for separate performance metrics but must work together to determine the common metric. Therefore, we adopted fixed linkage which makes either individual evolve in accordance with its counterpart, hence enables synergy between them.

Joint Fitness Evaluation. In GP, 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 individual where the fitness indicates the goodness of the candidate solution [30]. In this work, fitness is jointly determined by both bitrate and playback adaptation logics, so it should be evaluated upon the combined individual, i.e., (27).

This presents a challenge as the fitness of the adaptation logic is 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 according to a given fitness function (e.g., (6)). To ensure that the evaluation covers a broad range of network conditions, each combined individual is evaluated over L (e.g., $L = 200$) streaming sessions using throughput trace data. Now given the trace data of session j, denoted by S_j, 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, 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 objective function adopted (i.e., (6)).

Selection, Crossover, and Mutation. Once the fitness of 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 out the parent individuals with good fitness. Crossover/Mutation is to explore the combination/modification of the parent's genes such that the gene diversity in 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 of 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 logic for online streaming.

5.3 FLAS-GP

In this section, we apply FLAS to the above GP scheme. Specifically, in offline training, FLAS-GP uses state quantizer (SQ) to quantify $M \times N$ latency-QoE tradeoff states, i.e., (10). For each state, it executes a separate evolutionary process (c.f. Section 5.2), denoted by the function $T_{GP}(\cdot)$, to evolve a specialized adaptation logic:

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

where U_p denotes the objective function with latency coefficient $\boldsymbol{\omega}_p$ (defined in (6)), C_q denotes throughput trace data class q (defined in (9)), and $A_{p,q}$ is the evolved logic set including a total of $M \times N$ adaptation logics. It is worth noting that GP is only one candidate scheme to carry out FLAS's training phase, and other machine learning or heuristic paradigms can be operated in a similar manner.

Online Adaptation Logic Selection is identical to that in Section 4.2. In particular, the video player does not need any GP evolutionary components or expensive computational operations at runtime. The only modification needed is to append a lightweight module into the player to determine the state of each epoch online and then apply the matching logics 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.

5.4 Discussions on Online Training

FLAS trains the adaptation logics in an offline task, so the logic is generated a priori and then is unmodified after deployments. However, conceivably, the logics may still need to be re-trained upon major changes in the network environment, e.g., when introduced into 6G networks with much higher bandwidth limit. Therefore, the current two

TABLE 5
Actual Mean Latency (s) Vs. Target Latency

Algorithm	Target Latency (s)				
	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 6
Comparison of Latency-MAD (s)

Algorithm	Target Latency (s)				
	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

phase design (see Section 5.3) may not be flexible enough to deal with this situation.

Nevertheless, FLAS can inherently support an approach, i.e., conducting the training online directly on the video client, where the adaptation logic can be re-trained periodically as new data arrives. However, this raises two challenges. First, it significantly increases the computational overhead for the client. Second, it requires the system to learn from a small amount of data and converge to good performance quickly. We leave this as our future work.

6 Performance Evaluation

In this section, we conduct a systematic and thorough evaluation for FLAS and compare it to the state-of-the-arts. We employed the trace-driven simulations as described in Section 3. The configurations of FLAS are summarized in Table 3. For the GP evolutionary process, we adopted a population size of 800 (i.e., 800 combined individuals), and the population was evolved for 50 generations.

6.1 Latency-QoE Tradeoff

Among all the algorithms evaluated, three of them support targeting playback latency, namely FLAS-GP, FLAS-L2AC, and LAPAS. We first evaluate how well the three track the target latency. Table 5 summarizes the actual latency versus the target, and the results show that the three algorithms perform similarly, all of which achieve the actual latency close to the target.

We further quantify the deviation of the actual latency from the target, by defining a new metric – *Latency Mean Absolute Deviation* (henceforth called "*Latency-MAD*"), which characterizes the average absolute difference between the actual and the target latency:

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

where $\boldsymbol{\beta}_j$ is the actual mean latency during streaming epoch j, $\boldsymbol{\lambda}$ is the target latency and L is the total number of streamed epochs.

The results are summarized in Table 6 where lower latency-MAD indicates the target latency being better tracked. We can see that the two FLAS-based algorithms achieve significantly lower latency-MAD than LAPAS. This benefits from FLAS's online adaptation logic selection which periodically adjusts the operational logic to adapt to the changing network conditions.

Next, we investigate the tradeoff performance between QoE and playback latency. We observed in Fig. 10 that FLAS-GP achieves a continuous tradeoff trajectory where QoE is much higher than all other algorithms across the latency from 1.0s to 9.0s. Table 7 summarizes the QoE of FLAS-GP, FLAS-L2AC and LAPAS across throughput level 0∼9. Note that 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, and outperforms the other two algorithms significantly across all the throughput levels. This clearly demonstrates that with the flexible expression tree, GP enables FLAS to generate more specialized algorithms to match the network conditions better.

Table 8 compares the video quality (measured by the SSIM model from [39]), rebuffering duration, and video bitrate switches for the three algorithms. The results show that FLAS-L2AC exhibit substantially longer rebuffering duration and more bitrate switches, thereby degrading its QoE performance. This is largely due to FLAS-L2AC's aggressiveness in bitrate selection (c.f. Section 4.3) which is not suited in low bandwidth and high variability networks. In comparison, although FLAS-GP does not outperform other algorithms on every QoE metric, it is able to balance each metric to optimize the overall QoE.

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]:

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

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

$$\vartheta_k = \log\left(r_k / r_{\min}\right), \tag{33}$$

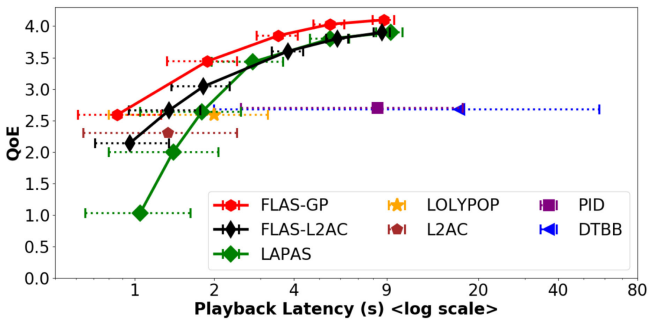


Fig. 10. Comparison of latency-QoE tradeoff under QoE function (1) (error bars span the streaming session with top/bottom 10 percent latency).

TABLE 7
QoE Performance Across Throughput Levels
(Target Latency = 2s)

Algorithm	Throughput Level				
	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

where r_k is the bitrate selected for segment k and $r_{\min}$ is the lowest available bitrate. The penalty weights in (32) follow [17] and [41]. Fig. 11 plots the latency-QoE tradeoff under this QoE function. We observed 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.

6.2 Robustness

In this section, we investigate the robustness of FLAS. Again, we only show the results under 2s target latency, as similar results were obtained with other options. We first consider temporal robustness. Fig. 12 plots the daily latency-MAD (defined by (31)) over a period of 40 days (the trace data is from #1, c.f. Table 2). 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 (see the daily throughput variations) over the 40 days. We also plotted the daily mean QoE, where FLAS-GP outperforms FLAS-L2AC and LAPAS substantially, achieving the highest QoE in 39 of the 40 days.

Next, we consider spatial robustness – performance over different network characteristics, e.g., network types, service providers and etc. (see Table 2). 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.

Tables 9 and 10 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 percent across the seven datasets, which again exhibits the superiority of the GP scheme. More interestingly, FLAS trained with "-D1" and "-Mixed" perform similarly across the seven datasets despite being trained

TABLE 8
Comparison of Streaming Metrics. (Target Latency = 2s)

	FLAS-GP	FLAS-L2AC	LAPAS
Video quality (SSIM)	0.949	0.956	0.922
Rebuffer duration (s)	9.5	21.3	12.6
Bitrate switches ($\times 10^2$)	5.6	13.4	4.4

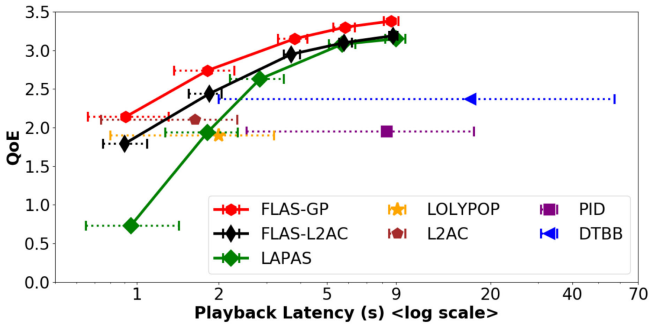


Fig. 11. Comparison of latency-QoE tradeoff under QoE function (32) (error bars span the streaming session with top/bottom 10 percent latency).

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.

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 adaptation logic 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, c.f. Section 5.4). This can greatly simplify the system deployment.

6.3 Sensitivity Analysis

In this section, we analyze the sensitivity of FLAS with respect to epoch duration and live event duration. FLAS executes the online adaptation logic selection at the beginning of each epoch (c.f. Section 4.2). This leads to the question of epoch duration choices. Table 11 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 logics 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 11, the latency-

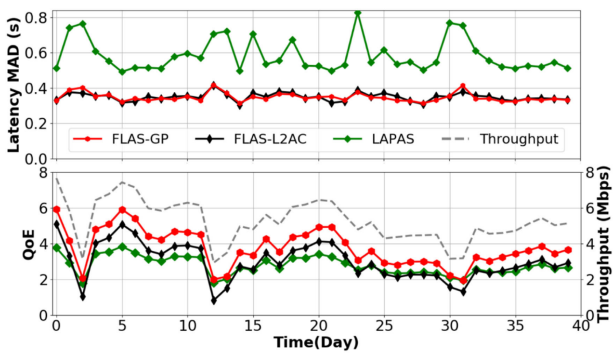


Fig. 12. Comparison of latency-MAD and QoE over 40 days (target latency = 2s).

TABLE 9
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

TABLE 10
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

MAD increases with longer epoch durations because longer epochs reduce the execution frequency of FLAS's online adaptation logic 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 we adopted 300s as the default epoch duration in this work.

Live events can have a very wide range of durations, ranging from minutes to hours, and another advantage of epoch-based FLAS is that the offline training can be decoupled from the actual live event duration. For example, Table 12 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 proposed in this paper offers a new approach to flexible latency 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 into the real streaming platforms, offering service providers a new tool for high-performance live streaming services.

TABLE 11
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 12
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

This work is only the first step in this direction. There are many opportunities for future research. For instance, as FLAS 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, in addition to DASH, there are also some other prevalent live streaming protocols, e.g., WebRTC [42], so exploring FLAS's extension to support different protocols is also a fruitful direction for future work.

Acknowledgments

The authors would like to thank the associate editor and the anonymous reviewers for their insightful comments in improving this article. This work was supported in part by the Centre for Advances in Reliability and Safety Limited (CAiRS), under AIR@InnoHK Research Cluster, General Program of National Natural Science Foundation of China under Grant 62072439, in part by the National Key Research and Development Program of China (13th Five-Year Plan) under Grant 2016YFB1000200, in part by the Shandong Provincial Natural Science Foundation under Grant ZR2019LZH004, in part by the Beijing Municipal Natural Science Foundation under Grant 4212028, and in part by the State Key Laboratory of Computer Architecture Innovation Fund under Grant carch4503.

References

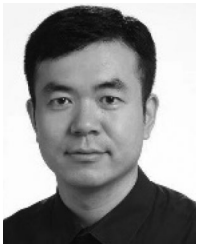
[1] YouTube Live. Accessed: Apr. 8, 2021. [Online]. Available: https://www.youtube.com/channel/UC4R8DWoMoI7CAwX8_LjQHig
[2] Facebook Live. Accessed: Apr. 8, 2021. [Online]. Available: https://live.fb.com/
[3] Latency Options of YouTube Live. Accessed: Apr. 8, 2021. [Online]. Available: https://support.google.com/youtube/answer/7444635?hl=en
[4] Latency Options of Twitch. Accessed: Apr. 8, 2021. [Online]. Available: https://help.twitch.tv/s/article/low-latency-video?language=en_US
[5] Latency Options of Amazon Web Services for Live Streaming. Accessed: Apr. 8, 2021. [Online]. Available: 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 Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1346–1358, Jun. 2020.
[7] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proc. ACM Conf. Multimedia Syst.*, 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," in *Proc. Conf. Emerg. Netw. Experiments Technol.*, 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," in *Proc. ACM SIGCOMM*, 2014, pp. 187–198.
[10] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. IEEE INFOCOM*, 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," in *Proc. ACM SIGCOMM*, 2015, pp. 325–338.
[12] A. H. Zahran, D. Raca, and C. Sreenan. "ARBITER+: Adaptive rate-based intelligent HTTP streaming algorithm for mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 12, pp. 2716–2728, Apr. 2018.
[13] Y. Qin *et al.*, "A control theoretic approach to ABR video streaming: A fresh look at PID-based rate adaptation," in *Proc. IEEE INFOCOM*, 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," in *Proc. IEEE INFOCOM.*, 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 Trans. Mobile Comput.*, vol. 16, no. 12, pp. 3488–3501, Dec. 2017.
[16] Z. Akhtar, Y. S. Nam, and R. Govindan, "Oboe: Auto-tuning video ABR algorithms to network conditions" in *Proc. ACM SIGCOMM*, 2018, pp. 44–58.
[17] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. ACM SIGCOMM*, 2017, pp. 197–210.
[18] F. Chiariotti, S. D'Aronco, and L. Toni, "Online learning adaptation strategy for DASH clients," in *Proc. ACM Multimedia Syst.*, 2016, pp. 8:1–8:12.
[19] V. Martín, J. Cabrera, and N. García, "Design, optimization and evaluation of a q-learning HTTP adaptive streaming client," *IEEE Trans. Consum. Electron.*, vol. 62, no. 4, pp. 380–388, Nov. 2016.
[20] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-DASH: A deep q-learning framework for DASH video streaming," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 703–718, Dec. 2017.
[21] N. Chen *et al.*, "Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 830–841, Apr. 2021.
[22] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proc. ACM Multimedia Syst.*, 2011, pp. 145–156.
[23] J. Wang, S. Meng, J. Sun, and Z. Quo, "A general PID-based rate adaptation approach for TCP-based live streaming over mobile networks," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2016, pp. 1–6.
[24] L. Xie, C. Zhou, and X. Zhang, "Dynamic threshold based rate adaptation for HTTP live streaming," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.
[25] K. Miller, A. K. Al-Tamimi, and A. Wolisz, "QoE-based low-delay live streaming using throughput predictions," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, pp. 1–24, Jan. 2017.
[26] M. Lim, M. N. Akcay, and A. Bentaleb, "When they go high, we go low: Low-latency live streaming Dash.js with LoL," in *Proc. ACM Multimedia Syst. Conf.*, 2020, pp. 321–326.
[27] Y. Zhao, Q. W. Shen, and W. Li, "Latency aware adaptive video streaming using ensemble deep reinforcement learning," in *Proc. ACM Int. Conf. Multimedia*, 2019, pp. 2647–2651.
[28] G. Zhang, and J. Y. B. Lee, "LAPAS: Latency-aware playback-adaptive streaming," in *Proc. IEEE Wirel. Commun. Netw. Conf.*, 2019, pp. 1–6.
[29] V. Mnih, A. P. Badia, M. Mirza, and A. Graves, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[30] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statis. Comput.*, vol. 4, no. 2, pp. 87–112, Jun. 1994.
[31] L. Golubchik, J. C. S. Lui, and R. R. Muntz, "Reducing I/O demands in video-on-demand storage servers," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 1995, pp. 25–36.
[32] T. Stockhammer, "Dynamic adaptive streaming over HTTP: Standards and design principles," in *Proc. ACM Multimedia Syst.*, 2011, pp. 133–144.
[33] Trace Driven Simulator. Accessed: Apr. 8, 2021. [Online]. Available: https://github.com/NGnetLab/Live-Video-Streaming-Challenge
[34] M. A. Potter, and K. A. D. Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.
[35] K. H. Ang, G. Chong, and Y. Li, "PID control system analysis, design, and technology," *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 4, pp. 559–576, Jul. 2005.
[36] Mobile Throughput Trace Data. Accessed: Apr. 8, 2021. [Online]. Available: http://sonar.mclab.info/tracedata/TCP/
[37] "Best practices for creating and deploying HTTP live streaming media for the iPhone and iPad," Apple Inc., Cupertino, CA, Aug. 2016. [Online]. Available: https://developer.apple.com/library/ios/technotes/tn2224/_index.html
[38] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. ACM Multimedia Syst.*, 2013, pp. 114–118.
[39] A. Bentaleb, A. C. Begen, R. Zimmermann, and S. Harous, "SDNHAS: An SDN-enabled architecture to optimize QoE in HTTP adaptive streaming," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2136–2151, Oct. 2017.
[40] T. Lyko, M. Broadbent, and N. Race, "Evaluation of CMAF in live streaming scenarios," in *Proc. ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2020, pp. 21–26.
[41] G. Yi *et al.*, "The ACM multimedia 2019 live video streaming grand challenge," in *Proc. ACM Int. Conf. Multimedia*, 2019, pp. 2622–2626.
[42] J. Kim, Y. Jung, and H. Yeo, "Neural-enhanced live streaming: Improving live video ingest via online learning," *Proc. ACM SIGCOMM*, 2020, pp. 107–125.
[43] W. Law, "Ultra-low-latency streaming using chunked-encoded and chunked-transferred CMAF," Akamai, Tech. Rep. 19, Oct. 2018.

Guanghui Zhang received the master's degree in electronic science and technology from Peking University, Beijing, China, in 2016 and the PhD degree in information engineering from The Chinese University of Hong Kong, Shatin, Hong Kong, in 2020. He is currently a postdoctoral fellow with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong, where he participated in the research and development of multimedia technology.

Jack Y. B. Lee (Senior Member, IEEE) received the BEng and PhD degrees in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 1993 and 1997, respectively. He is currently an associate professor with the Department of Information Engineering, Chinese University of Hong Kong. His research group focuses on multimedia communications systems, mobile communications, protocols, and applications. He specializes in tackling research challenges arising from real-world systems. He works closely with the industry to uncover new research challenges and opportunities for new services and applications. His systems research from his lab have been adopted and deployed by the industry.

Ke Liu received the BEng and PhD degrees in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 2008 and 2013, respectively. From 2017 to 2018, he was a postdoc scholar with the School of Industrial Engineering, Purdue University, IN, USA. He is currently an associate professor with the Advanced System Group, Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Science, Beijing, China, where he participated in the research of protocol optimization and cloud computing.



Haibo Hu (Senior Member, IEEE) is currently an associate professor with the Department of Electronic and Information Engineering, Hong Kong Polytechnic University, and the programme leader of BSc (Hons.) in information security. He has authored or coauthored more than 80 research papers in refereed journals, international conferences, and book chapters. His research interests include cybersecurity, data privacy, Internet of Things, and adversarial machine learning. As a principal investigator, he was the recipient of more than 20 million HK dollars of external research grants from Hong Kong and mainland China as of year 2020. He was on the organizing committee of many international conferences, such as ACM GIS 2020 and 2021, IEEE ICDSC 2020, IEEE MDM 2019, DASFAA 2011, DaMEN 2011 and 2013 and CloudDB 2011, and on the programme committee of dozens of international conferences and symposiums. He was the recipient of a number of titles and awards, including the IEEE MDM 2019 Best Paper Award, WAIM Distinguished Young Lecturer, ICDE 2020 Outstanding Reviewer, VLDB 2018 Distinguished Reviewer, ACM-HK Best PhD Paper, Microsoft Imagine Cup, and GS1 Internet of Things Award.

Vaneet Aggarwal (Senior Member, IEEE) received the BTech degree in electrical engineering from the Indian Institute of Technology Kanpur, India, in 2005, and the MA and PhD degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2007 and 2010, respectively. From 2010 to 2014, he was a senior member of the Technical Staff Research, AT&T Labs Research, Bedminster, NJ, USA. He was an Adjunct Assistant Professor with Columbia University, NY, from 2013 to 2014, and an Adjunct Professor with IISc Bangalore, India, from 2018 to 2019. He is currently a faculty with Purdue University, West Lafayette, IN, USA. His current research interests include communications and networking, video streaming, cloud computing, and machine learning. Dr. Aggarwal was the recipient of the Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, the AT&T Vice President Excellence Award in 2012, the AT&T Senior Vice President Excellence Award in 2014, the 2017 Jack Neubauer Memorial Award recognizing the Best Systems Paper published in the *IEEE Transactions on Vehicular Technology*, and the 2018 Infocom Workshop HotPOST Best Paper Award. He is on the Editorial Board of the *IEEE Transactions on Communications*, *IEEE Transactions on Green Communications and Networking*, and *IEEE/ACM Transactions on Networking*.